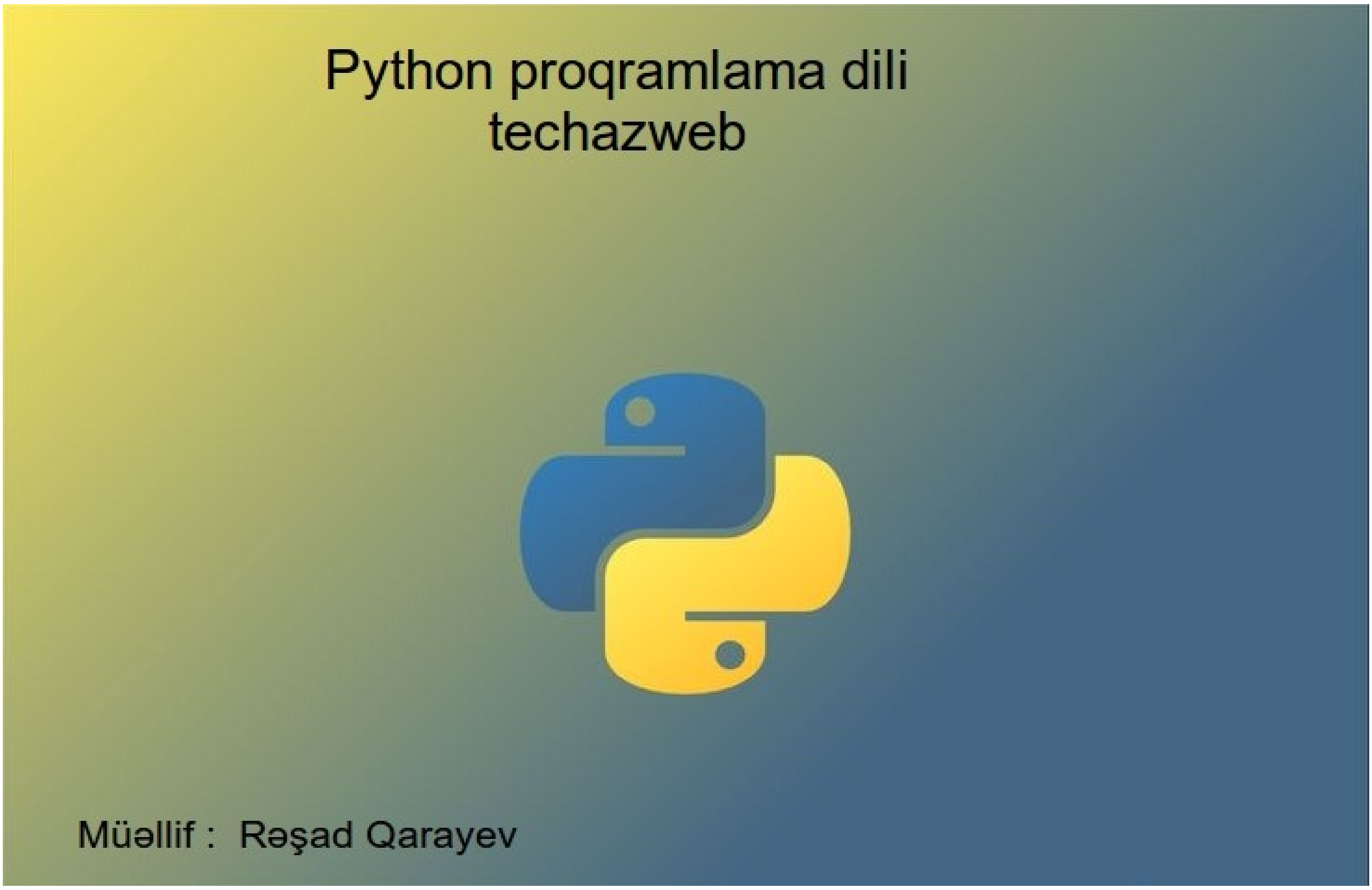


Python 3...

Müəllif Rəşad Qarayev

Mündəricat

Kitabı yükləmək üçün

https://techazweb.wordpress.com/e-book/

Kitabla bağlı rəylərinizi

pythonaz@yahoo.com

https://techazweb.wordpress.com/əlaqə/

ünvanlarına bildirə bilərsiniz.

Python3-ə keçməmişdən öncə python2-i oxumağınızı təklif edirəm. python 2-dən başlayın,daha sonra python3, sizə çox rahat gələcək.Hal-hazırda bir çox paketlər python 2 versiyası üzərində yazılıb.python3-ə keçid hələ biraz zaman alacaq.
Python paketlərini yükləmək üçün

```
sudo apt-get install python-pip
sudo easy_install  pip
easy_install  paket_adı
pip   install   paket_adı
pip   install   paket_adi==version
```

Python demək olarki bütün əməliyyat sistemlərində çalışır.

Terminalı açırıq

```
$ python3 -V
```

Python 3.4.3

və sistemdə deməli python 3.4.3 versiyası yüklüdür.Əgər yüklü deyilsə o zaman *sudo apt-get install python3* yazaraq sisteminizə yükləyə bilərsiniz.

Windows istifadəçiləri isə pythonu http://www.python.org/downloads ünvandan yükləyə bilər.

Mac OS istifadəçiləri
https://www.python.org/downloads/mac-osx/

CentOS və Red Hat server istifadəçiləri

```
$ su
$ sudo yum install yum-utils
$ sudo yum-builddep python
$ curl -O https://www.python.org/ftp/python/3.5.0/Python-3.5.0.tgz
$ tar xf Python-3.5.0.tgz
$ cd Python-3.5.0
$ ./configure
$ make
$ sudo make install
```

Sistemimizə python3 -ü xətasız yüklədikdən sonra terminalı açırıq və ardından

```
$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

>>>

python3 çağırdığımızda terminalda yazıldığı kimi biz Python 3.4.3 versiyası üzərindəyik.

Help() yazaraq pythondan yardım istəyə bilərik

```
>>> help()

Welcome to Python 3.5's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/3.5/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".
```

Kitabxanalar haqqında məlumat almaq üçün python-sətrinə ifadənizi yazıb enter(daxil et)   düyməsini basın

```
help> pypy
No Python documentation found for 'pypy'.
Use help() to get the interactive help utility.
Use help(str) for help on the str class.

help>
```

və quit yazmaqla python shell-ə qayıda bilərik.

```
help> quit
```

You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)". Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
>>>

Fikir bildirmək

pythonda kodlarınızı yazarkən çalışın yazdığınız kodun qarşısında açıqlamasın da verin.Bu həm sizin üçün,həmdə paylaşacağınız bir programı başqaları üçün fikriniz anlaşılan olsun.
Rəy bildirmək üçün python bizə #-işarəsini təklif edir

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
name=input('write you name:')# istifadəçidən ad soruşaq
surn=input('write you surname:')# istifadəçidən soyadı soruşaq
print('name {1} and surname {0} '.format(name,surn))

write you name:asd
write you surname:ert
name ert and surname asd
>>>
```

Yaşıl rəngdə olan ifadələrimiz, kodlar haqqında rəydir.rəy bildirmək üçün # -işarəsindən istifadə etdik.

pythonda sabit ifadələr

```
>>> from keyword import*
>>> print(kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def',
'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in',
'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while',
'with', 'yield']
>>> len(kwlist)
33
```

```
>>>
```

deməli python3-də 33 sabit ifadə var.

Python-a giriş

print() funksiyası

print () funksiyası ilə başlayacağıq.print() artıq python3-də funksiya olaraq tanınır.
Ala bildiyi metodlar
sep,end,file,flush

```
>>> print 'texnika'
  File "<stdin>", line 1
    print 'texnika'
                  ^
SyntaxError: Missing parentheses in call to 'print'
>>>
```

Yuxarıda print 'texnika' yazaraq xəta aldıq.Çünki biz python3-ü istifadə edirik.python3-də print() funksiyası , print('ifadə') şəklində yazılır.

```
>>> print('algebra')
algebra
>>>
```

Və heç bir xəta almadıq.və ya

```
>>> 'algebra'
'algebra'
>>>
```

print() funksiyasından istifadə etməyərək dırnaq içində ifadəni ekrana çap etdik.
Eləcədə başqa bir ifadə ilə qeyd edib çap edə bilərik.

```
>>> i='algebra'
>>> print(i)
```

```
algebra
>>>
```

Dırnaq işarələri

python 2-də olduğu kimi python3-də də eyni qayda ilə dırnaq işarələrindən istifadə olunur.Tək,cüt və üçəm dırnaq.Sadəcə ifadə və dırnaq işarələri mötərizənin daxilində yazılır.

```
>>> print('techaz')
techaz
>>> print("techaz")
techaz
>>> print("""techaz""")
techaz
>>>
```

Sadəcə python2-dən fərqli olaraq python3-də tək dırnaq içində cüt dırnaqdan da istifadə etmək olur.

```
>>> print('hello"linux"hello')
hello"linux"hello
>>>

>>> print('hello"""linux"""hello')
hello"""linux"""hello
>>>
```

python3-də print () funksiyası, daxilində bir neçə ifadə almaq imkanına malikdir.

```
>>> print('Ali','Natasha','Eldar')
Ali Natasha Eldar
>>>
```

sayları isə dırnaq içində göstərməyə məcbur deyilsiniz.

```
>>> print ('Ali','Natasha','Eldar',2016)
Ali Natasha Eldar 2016
>>>
```

print () funksiyasının ala bildiyi metodlar

sep metodu

Bu metod ifadələr arasına istənilən işarələr,eləcədə kəlimələr,hərflər artıra bilir.

Yuxarıda yazdığımız ifadələrə ('Ali','Natasha','Eldar',2016) fikir versəniz ekran çapından sonra hər birinin arasına boşluq ataraq çap etdi.Hər dəfə biz dırnaq daxilində vergül,nöqtə qoymaqdan yorulmayaq deyə bizə sep metodu kömək edəcək.

```
>>> print('Ali','Natasha','Eldar',2016,sep=',')
Ali,Natasha,Eldar,2016
>>>
```

Gördüyünüz kimi sep=',' metodu vasitəsilə ifadələr arasına vergül qoy əmri verdik.
Eyni qayda ilə

```
>>> print('www','kapitalbank','com',sep='.')
www.kapitalbank.com
>>>
```

yazaraq bir adresi tamamlamış olduq.

```
>>> print(1,2,3,4,sep='.hello')
1.hello2.hello3.hello4
>>>
```

Əgər sep metodundan istifadə edərək ifadələr arasına boşluq qoymaq istəsəz,o zaman None ifadəsindən istifadə edin.

```
>>> print('Kharabag','Shusha','Baki',sep=None)
Kharabag Shusha Baki
```

```
>>>
```

sep metodu yalnız saylar qəbul etmir.

```
>>> print('Kharabag','Shusha','Baki',sep=1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sep must be None or a string, not int
>>>
```

Xətadan gördüyümüz kimi – sep metodu yalnız None və string parametrləri alır,saylar deyil.

sep metodu xüsusi işarələri də daxilində istifadə edir.

```
>>> print('Hello','world',sep='\n')
Hello
world
>>>
```

end metodu

Bu metod dilimizə ‘davamı,ardı’ kimi tərcümə olunur.Və pythonda da mənası qədər iş görür.

```
>>> print('hello world',end='.')
hello world.>>>
>>>
```

Yuxarıda kodumuzu çalışdırdıqda qarşımıza ifadə sonunda ps1 işarəsinin çıxdığını görürük.Bu metodu istifadə etdikdə nöqtədən əvvəl xüsusi işarəmiz olan \n -dən istifadə edəcik.

```
>>> print('hello',end='.\n')
hello.
>>>
```

Və ifadənin sonuna nöqtə qoymağa müvəffəq olduq

sep metodunda olduğu kimi,end metodu da None parametrini ala bilir.

```
>>> print('hello','python',end=None)
hello python
>>>
```

Eyni qayda ilə end metodu da sayları ala bilmir.

```
>>> print('hello',end=0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: end must be None or a string, not int
>>>
```

file metodu File(I/O)

(fayl=open("techaz.txt", "w")
İstifadəsi isə mütləq açacağınız faylı bir ifadəyə atıb qeyd etməkdir.misallara baxaq.

```
>>> fayl=open('techaz.txt','w')
>>> print('techaz',file=fayl)
>>> fayl.close()
```

```
>>>
```

Kodlarımızda ilk əvvəl bir techaz.txt faylını write modunda açdıq,daha sonra fayla əlavə edəcəyimiz ifadəni yazıb,file metodundan istifadə edərək faylı çağırdıq və ardından faylımızı fayl.close() yolu ilə bağladıq.Əgər sonda fayl.close() yazmasaq ifadəmiz fayla daxil olmayacaq.Bunu biz python2-dən bilirik.

flush metodu

bu metod file.close() ifadəsini əvəz edərək,faylı bağlamadan ifadələrinizi fayla yaza bilərsiniz.Aldığı parametrlər True (yəni təsdiq) , False(yəni inkar)
Yuxarıda yazdığımız fayl kodlarını təkrar yazaq.

```
>>> fayl=open('techaz.txt','w')
>>> print('texas',sep='\n',file=fayl,flush=True)
>>>
```

Kodlarımızda tək yenilik olan flush=True ifadəsi oldu.Və fayl.close() yazmadan flush=True ifadəsi ilə faylımıza texas sözünü yazdıq.

Pythonda xüsusi işarələr

Bu bəhsdə keçəcəyimiz xüsusi işarələri aşağıdakı cədvələ qeyd edək

\	Apastroflu ifadələrdə istifadə olunur
\n	Ifadəni növbəti sətrə keçidini təmin edir
\\	Digər xüsusi işarələri,pythonun

	görməməsini təmin edir
\t	Ifadəni bir tab irəli atır
\u	Qarşılıq gələn unikodlar(4 saydan ibarət)
\U	Qarşılıq gələn unikodlar(8 saydan ibarət)
\N	Hər bir hərfin unikodda ifadə qarşılığı
\a	Siqnal səsi
\r	Öncəki ifadəni silərək özündən sonra gələn ifadəni ilk ifadə üzərinə yazır.
\v	Sonrakı ifadəni alt sətrə keçirərək bir tab irəli atır
\b	Ilk ifadənin son hərfini silir
r	Xüsusi işarələri string cins tipində göstərir.

tərs əyri xət ('\')

```
>>> print('Hello lin\'uks hello')
Hello lin'uks hello
>>>
```

bu parametrdən apastroflu ifadələrdə istifadə edə bilərsiniz.
Eləcədə eyni dırnaq işarələrinin bir mötərizə daxilində istifadəsinə yardımçı olur.

```
>>> print('\'Nikola Tesla \'haqqında bir çox kitab yazılmışdır.')
'Nikola Tesla 'haqqında bir çox kitab yazılmışdır.
>>>
```

Yuxarıda gördüyümz kimi Nikola Tesla ifadəsini xüsusi ad kimi qeyd etməyimiz üçün əvvəlinə tək-dırnaq içində \ tərs əyri xətt və dırnaqdan istifadə etdik.

(\n) xüsusi işarəsi

Biz bu xüsusi işarəyə python2-dən tanışıq.Bildiyimiz kimi sətrin başında və sonunda fərqli nəticələr verə bilir.

```
>>> print('Hello\n','world')
Hello
world
>>> print ('Hello world\nSalam dünya')
Hello world
Salam dünya
>>>
```

İki tərs əyri xətt (\\)

Bu metoda gəlin bir misalla baxaq.

```
>>> print('C:\ninja.txt')
C:
inja.txt
>>>
```

Kodlarımızda gördüyünüz kimi C-qovluqundakı ninja.txt faylına keçid almaq istədik amma \n xüsusi işarə olduğundan python onu bir kənara qoyub inja.txt faylı kimi çap etdi.Bunun üçün biz iki tərs əyri xətdən istifadə edərək bu çətin işin öhdəsindən gələcəyik.

```
>>> print('C:\\ninja.txt')
C:\ninja.txt
>>>
```

\t tab işarəsi

Bu metod ifadələri bir tab düyməsi qədər(və ya 4 dəfə space düyməsi) irəli atır.

```
>>> print('\tFrankeynsteyn')
	Frankeynsteyn
>>> print('Stive\tJobs')
Stive	Jobs
>>>
```

Bu metodu sep parametri ilə də istifadə edə bilərik

```
>>> print('1','2','3','4',sep='\t')
1	2	3	4
>>>
```

və ya

```
>>> print('1','2','3','4',end='\t')
1 2 3 4	>>>
>>>
```

Siqnal səsi (\a) işarəsi

Bu metod əsasən windows əməliyyat sistemində çalışır.Linux sistemində isə işarəsini alarm olaraq çap edir.

```
>>> print('\a'*10)

>>>
```

\r xüsusi işarəsi

Bu işarə bir neçə ifadə ilə özünü biruzə verir.Beləki print() vasitəsilə bir neçə ifadəni ekrana çap edək.

```
>>> print('John Ellidor')
John Ellidor
>>>
```

daha sonra \r işarəsini hər hansı bir yerə yerləşdirib çap edək

```
>>> print('John\rEllidor')
Ellidor
>>>
```

Yuxarıda gördüyümüz kimi John sözünü ekrana çap etmədi.bu işarə bir növ delete funksiyasını yerinə yetirir.İndi son kodumuza izah verək.
İlk öncə \r işarəsi mötərizə daxilində olan ilk ifadə John-u ekrana yazır.Daha sonra \r işarəsindən sonra gələn sözü (Ellidor) həmin John ifadəsinin üzərinə yazır.Amma yazarkən hərf sayını gözdən keçirir,əgər birinci ifadə ikinci ifadədən kiçikdirsə o zaman ikinci ifadə tamamı ilə ekrana yazılacaq.Yox əgər ilk ifadə hərf sayı çoxdursa ozaman ikinci ifadə ilə bərabərləşən hərflərdən başqa ilk ifadənin yerdə qalan hərfləri ikinci ifadənin sonuna əlavə olunaraq ekrana çap olunacaq.
Yəni

J	o	h	n	-	-	-
E	l	l	i	d	o	r

gördüyünüz kimi john 4-hərfdən ibarətdir.Yəni Elli ,John sözünü tamamilə silə bilir.İndidə ilk ifadənin daha çox hərf oduğu kodlarımıza baxaq.

```
>>> print('Ellidor\rJohn')
Johndor
>>>
```

Və burda hər şey aydın oldu.John 4-hərfdən ibarət olduğu üçün ilk Elli ifadəsini sildi,yerdə qalan dor hərflərini də öz üzərinə gəldi.

```
>>> print('Elli\rdor John')
dor John
>>>
```

\v işarəsi

bu işarə özündən sonra gələn ifadəni aşağı sətrə keçir və sətir başından bir tab irəli atır.

```
>>> print('Elmler\vakademiyasi')
Elmler
      akademiyasi
>>>
```

(\b) işarəsi

Bu işarə özündən əvvəl gələn ifadədən bir sıra silir və özündən sonra əgələn ifadəni öncəki ifadə ilə birləşdirir.

```
>>> print('algebra\b+')
algebr+
>>>
```

Kodlardan gördüyümüz kimi \b işarəsi a hərfini silərək özündən sonra gələn + işarəsini algebr ifadəsinə birləşdirdi.

```
>>> print('kapitall\bbank.com')
kapitalbank.com
>>>
```

Eyni qayda ilə kapitall ifadəsindən l-hərfin birini silərək özündən sonra gələn bank.com ifadəsini kapital sözü ilə birləşdirdi.və nəticədə bir adresi doğru əldə etdik.

Unicode (\u) işarəsi

\n işarəsində olduğu kimi ana qovluğunuzun baş hərfi u ilə başlayarsa bu işarə istifadə olunduqda xəta ilə qarşılaşacağıq.

```
>>> print('C:\user\panda\techaz.txt')
  File "<stdin>", line 1
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated \uXXXX escape
>>>
```

Və öncə öyrəndiyimiz kimi \u işarəsini ya əyri xətlə yada ikili tərs əyri xətlə qeyd edin

```
>>> print('C:/user/panda/techaz.txt')
C:/user/panda/techaz.txt
>>>
```

və ya

```
>>> print('C:\\user\panda\techaz.txt')
C:\user\panda	echaz.txt
>>>
```

İndidə gördüyünüz kimi bizim faylımız t hərfi ilə başladığından python \techaz ifadəsində ilk \t ni xüsusi işarə olaraq gördü.Bunun üçündə hər zaman belə kodlarda print('C:/user/panda/techaz.txt') ifadəsində olduğu kimi işarədən istifadə etsək daha məqsədəuyğun olar.

Hərflərin sayı bazası,unikodlarla

Bildiyimiz kimi hər bir hərfi bir rəqəm və ya rəqəmlər birləşməsi təmsil edir.
http://unicode-table.com/ru/#cyrillic ünvana daxil olaraq hər bir hərfin təmsil etdiyi compleks birləşmələri görə bilərsiniz
Bir neçəsinə dair misal çəkək.

```
>>> '\u0070'
'p'
>>> '\u0080'
'\x80'
>>> '\u00E0'
'à'
>>>
```

yazılış qaydası kodlarımızda olduğu kimi dırnaq içində tərs əyri xətt və ardından \u hərfi və cədvəldə olan sabit birləşmə yazılır.

Böyük unicode (\U)

işarə kiçik \u unikodla eyni işi yerinə yetirir sadəcə olaraq \u bütünlüklə 4 sayda olduğu halda \U isə tamamı 8 sayda olacaq,əvvəlinə 4-ədəd 0 rəqəmini daxil edəcik.

```
>>> '\U000000E0'
'à'
>>>
```

```
>>> '\U00000070'
'p'
>>>
```

(\N)   işarəsi

Əvvəl gəlin unicodedata modulunu çağıraraq hər bir hərfə qarşılıq gələn ifadəni tapaq

```
>>> import unicodedata
>>> unicodedata.name('ş')
'LATIN SMALL LETTER S WITH CEDILLA'
>>>
```

İlk əvvəl unicodedata modulunu çağırdıq sonra ş hərfinin unicode sistemində ifadəsini soruşuduq və bizə 'LATIN SMALL LETTER S WITH CEDILLA' ifadəsini göstərdi.İndi isə \N ifadəsinin bu unicodedata ifadələri ilə nə əlaqəsi olduğuna baxaq.
Bu işarə bir növ geri çevirmə işini yerinə yetirir.Yəni,biz unicodlarda olan hərflərin ifadəsini tapırıqsa indidə bu işarə vasitəsilə ifadələri soruşaraq hansı hərflərə uyğun gəldiyini tapaq.

```
>>> '\N{LATIN SMALL LETTER S WITH CEDILLA}'
'ş'
>>>
```

Və kodlarımızdan aydın olduki biz ifadə vasitəsilədə geri dönərək hansı hərfə uyğun gəldiyini soruşa bildik.

Bu xüsusi işarələrlə windows əməliyyat sistemində işləyərkən bir çox xətalar ala bilərsiniz.Odurki yuxarıda göstərdiyimiz \\ , / işarələrdən yerində düzgün istifadə edin.

Hexadecimal (\x) işarəsi

Onaltılıq say sistemlərində qarşılıq ifadələri göstərir.Aşağıdakı ünvana daxil olaraq
http://www.ascii.cl/ ordakı hex(onaltılıq say sistemi) sırasındakı qarşılıqlı rəqəmləri görə bilərsiniz.

Bir neçəsinə dair misal çəkək

```
>>> '\x45'
'E'
>>> '\x60'
'`'
>>> '\x40'
'@'
>>>
```

r   xüsusi işarəsi

Bu işarə mühüm işarələrdən biridir.Və pythonda digər işarələr kimi dırnaq daxilində deyil,xaricində istifadə olunur.Biz \n \t \u işarələrini keçdikdə qovluqlarda,ifadələrdə necə bizə problem yaratdığının şahidi olurduq.Bunlardan qaçmaq üçün biz ya \\ və ya / işarələrindən istifadə etməli olurduq.

```
>>> print('C:\ninja.txt')
C:
inja.txt
```

Yuxarıda gördüyünüz ninja.txt adlı faylımızı inja.txt kimi göstərdi və \n işarəsi aktivləşərək faylımızı aşağı sətrə atdı.İndi gəlin dırnaqdan əvvəl r ifadəsini istifadə edərək nəticələrə baxaq.

```
>>> print(r'C:\ninja.txt')
C:\ninja.txt
>>>
```

Və gördüyümüz kimi C -qovluğundakı faylımız öz adı ilə ekrana çap olundu.

Və ya

```
>>> print('C:\users\nonstop\techaz')
  File "<stdin>", line 1
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated \uXXXX escape
>>>
```

gördüyümüz kimi \u işarəsi başda olduğu üçün xəta aldıq.Sətrin əvvəlinə r hərfi atsaq

```
>>> print(r'C:\users\nonstop\techaz')
C:\users\nonstop\techaz
>>>
```

nəticəni doğru əldə edərik.

input()   funksiyası

Bu funksiya bizə python2-dən məlumdur.python2 də olan input və raw_input() funksiyaları python 3-də də eyni işi yerinə yetirir.Sadəcə python3-də raw_input() funksiyası qaldırılmışdır.Onun yerinə input funksiyasından istifadə olunur.

```
>>> print(raw_input('write you name:'))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'raw_input' is not defined
```

və xəta aldıq.

Amma

```
>>> print(input('write you name:'))
write you name:Python
Python
```

```
>>>
```

input funksiyasından rahatlıqla istifadə edə bildik.
input() funksiyası python3 də modulları bağladığımız kimi sonda istifadə oluna bilir.
Bir mətn faylı açırıq   sonunu py qoyuruq.Mən fr.py olaraq adlandırdım və ilk sətrə

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
i=input('write you name:')
print (i)
input()
```

yazaraq mətni bağlayırıq.

Daha sonra terminalı açırıq

```
techaz~$ ls
Desktop    Downloads  Pictures  sources.list  Templates
Documents  Music      Public    techaz.txt    Videos
panda@panda:~$ cd Desktop
panda@panda:~/Desktop$ ls
bluetooth.py   checkbuton.py   fr.py   maxresdefault.jpg
bluetooth.pyc  checkbuton.pyc  fr.pyc
panda@panda:~/Desktop$ python3 fr.py
write you name:rashad
rashad

techaz:~/Desktop$
```

Və gördüyümüz kimi terminaldan proqramımızı çalışdırdıq amma input funksiyası başda bizdən adımızı yazmağı soruşdu,daha sonra print funksiyası onu ekrana çap etdi və sonra proqramımız sanki açıq qalmış kimi özünü biruzə verdi,o zamana qədərki biz enter düyməsini basaq.

input() funksiyasına dair bir neçə proqramlar yazaq.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
name=input('write you password->')
if len(name)<8:
    print (name,'parolunuz qısadır')
else:
    print('parolunuz doğrudur')

>>>
write you password->panda
panda parolunuz qısadır
>>>
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
name=input('write you password->')
if len(name)<8:
    print (name,'parolunuz qısadır')
else:
    print('parolunuz doğrudur')


>>>
write you password->almanax1234
parolunuz doğrudur
>>>
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
print("""Nyuton qanununa görə F=ma,yəni qüvvə kütlə ilə/
təcilin hasilinə bərabərdir.Təcil beynəlxalq sistemdə /
a=9.8 m/san**2 olaraq qəbul edilmişdir\n""")
tecil=9.8
m=input('cismin kütləsin yazın:')
cavab=int(m)*int(tecil)
print('cismin kütləsi','\n',m+'kg')
print('cavab=',cavab,'N')
```

```
>>>
Nyuton qanununa görə F=ma,yəni qüvvə kütlə ilə/
təcilin hasilinə bərabərdir.Təcil beynəlxalq sistemdə /
a=9.8 m/san**2 olaraq qəbul edilmişdir

cismin kütləsin yazın:12
cismin kütləsi
 12kg
cavab= 108 N
>>>
```

Cins dəyişdirmələri

python2 də olduğu kimi python3-də də cins dəyişdirmələri eynidir.
Aşağıdakı cins dəyişdirmələrini nəzərdən keçirək.
int() str() float()   complex()

int()

int() cinsi,integer sözünün qısaltmasıdır.ingilis dilindən tərcümədə tam ədəd kimi tərcümə olunur.

int cinsi hesablamalar zamanı ən çox istifadə edəcəyimiz cins tipidir.Siz bir string cinsinə məxsus ifadəni integer,yəni tam sayıya çevirməniz üçün

```
>>> i='12'
>>> type(i)
<class 'str'>
>>> int(i)
12
```

kimi yaza bilərik.

integer cinsinə dair misallara baxaq.

```
>>> i='12'
>>> i*3
'121212'
>>> i**2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
>>>
```

və qüvvətə yüksəltmədə xəta ilə qarşılaşdıq,çünki bir string cinsinə mənsub ifadəni qüvvətə yüksəltmək olmaz birbaşa.Bunun üçün

```
>>> int(i)**2
144
>>>
```

int() cinsi vasitəsilə i-string cinsini integer cinsinə çevirdik,ardından qüvvətə yüksəltdik.

```
>>> type(int(i))
<class 'int'>
>>>
```

Tipini soruşduqda bizə integer cinsi olduğunu söylədi.

Python2 də biz string cinsləri üçün raw_input(),hesablama işlərində isə input() funksiyasından istifadə edirdik.Bildiyimiz kimi python3 də raw_input() funksiyası qaldırıldığından bu hər iki funksiyanın vəzifəsini tək input() funksiyası yerinə yetirir.input() funksiyası string cinsini ifadə edir,integer cinsi üçün isə biz cins dəyişmələrindən istifadə edirik.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
sual=input('rəqəm yazın:')
sual_1=input('ikinci rəqəm yazın:')
print ('cavab=',sual+sual_1)
```

```
rəqəm yazın:12
ikinci rəqəm yazın:12
```

```
cavab= 1212
>>>
```

ekran görüntüsündən nələrinsə tərs getdiyinin fərqindəyin.12+12=24 olması əvəzinə cavabda 1212 çıxdı.Yuxarıda qeyd etdiyim kimi əgər int cinsinə çevirmə etməsək,ilk başdan python input() funksiyasına daxil olan istənilən ifadəni string cinsində görəcək.

Qısa bir dəyişiklik edərək

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
sual=int(input('rəqəm yazın:'))
sual_1=int(input('ikinci rəqəm yazın:'))#integer cinsinə çevirdik
print ('cavab=',sual+sual_1)
```

```
rəqəm yazın:12
ikinci rəqəm yazın:12
cavab= 24
>>>
```

str(string) cinsi.

Bu cins tipi string sözünün qısaltmasıdır.input() funksiyasında qeyd etdiyimki kimi funksiya başdan ifadəni string cinsində görür.Amma bəzən bizə integer cinsindən string cinsinə keçid etmək lazım olur.Bunun üçün str() cins dəyişdirilməsindən istifadə edəcəyik.

```
>>> a='34'
>>> type(a)
<class 'str'>
```

```
>>> type('i')
<class 'str'>
```

```
>>>

>>> i=24
>>> type(i)
<class 'int'>
>>> int(i)
24
>>> type(i)
<class 'int'>
>>> str(i)
'24'
>>> type(i)
<class 'int'>
>>> type(str(i))
<class 'str'>
>>>
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
sual=str(input('adınızı yazın:'))
sual_1=str(input('soyadınızı yazın:'))#integer cinsinə çevirdik
print (sual,' ',sual_1)
```

```
adınızı yazın:Kamil
soyadınızı yazın:Habibov
Kamil   Habibov
>>>

adınızı yazın:1234
soyadınızı yazın:anonim
1234   anonim
>>>
```

Kodlarımızda str() cins dəyişdirməsindən istifadə etdikki,hər ehtimala qarşı istifadəçi rəqəm daxil edərsə onu string cinsinə çevirək.

float() cinsi.

Dilimizə kəsirli sayılar kimi tərcümə olunur.python2-dən fərqli olaraq python3-də kəsirli sayıları rahatlıqla bölə bilərsiniz və cavabı da dəqiqliklə alacaqsınız.

```
>>> 12.4/3
4.133333333333334
>>> type(12.4)
<class 'float'>
>>>
```

tam ədədi float() cinsində görmək üçün

```
>>> 12
12
>>> float(12)
12.0
>>> float(4.0)
4.0
>>> float(34)
34.0
>>>
```

və ya

```
>>> i='45'
>>> float(i)
45.0
>>>
```

complex() cinsi

kompleks cinsi dilimizə qarışıq kimi tərcümə olunur.Yəni kompleks ədədlər.

```
>>> i='45'
>>> float(i)
45.0
>>> complex(12)
(12+0j)
>>> complex(i)
(45+0j)
>>> complex('i')
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    complex('i')
ValueError: complex() arg is a malformed string
>>> complex(2)
(2+0j)
>>> complex(0)
0j
>>> type(2+0j)
<class 'complex'>
>>>
```

format() metodu

```
>>> print('{} və {} proqramlaşdırma dilidir.'.format('Python','Java'))
Python və Java proqramlaşdırma dilidir.
>>>
```

Yuxarıda gördüyünüz kodlarda format() metodundan istifadə edərək ardıcıl olaraq Python və Java ifadələrini (proqramlaşdırma dilidir)ifadəsindən önə gətirdik.

```
     Python      Java
        |          |
print('{}    və   {}  proqramlaşdırma dilidir.')
```

əgər biz cəm-mötərizəni boş yazarsaq o zaman format metodu daxilində yazılan ifadələr 1-ci ilk cəm mötərizəyə,2-ci isə ikinci cəm mötərizəyə uyğun gələcək.Və ya

0 1

```
>>> print('{1} və {0} proqramlaşdırma dilidir.'.format('Python','Java'))

Java və Python proqramlaşdırma dilidir.
>>>
```

yazarsaq,format metodu daxilindəki ifadələr tərs,1-ci Java 2-ci isə Python çap olunacaq.İfadələri artırmaq da olar

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
i=str(input('proqramlama dili yazın:'))
v=str(input('daha birin yazın:'))
x=str(input('növbəti proqramlama dili yazın:'))
print('{2},{0} və {1} dilləri,yüksək səviyyəli dillərdir.'.format(i,v,x))
```

```
proqramlama dili yazın:Python
daha birin yazın:Java
növbəti proqramlama dili yazın:C++
C++,Python və Java dilləri,yüksək səviyyəli dillərdir.
>>>
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import urllib.request
response=urllib.request.urlopen('www.python.org')
php=response.readlines()
print('{} adress'.format(php))
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import urllib.request

response=urllib.request.urlopen('http://www.open.az')
html=response.readlines()
for i in html:
```

```
    if i:
        print('{} adress'.format(i))
```

Əgər print() funksiyadan istifadə etməsək

```
>>> '{} and {}'.format('Multi','USB')
'Multi and USB'
>>>
```

Buraya qədər düşünürəm format() metodunun qaranlıq tərəfi qalmadı.

if ,else,elif   operatorları

if operatoru

Dilimizə ‘əgər’ kimi tərcümə olunur.
Bu operator bir çox proqramlama dillərində var.Əgər siz python 2-i oxumusuzsa sizə bu operatorlar çətin gəlməyəcək.Eyni funksiyaları ilə python3-də də eyni baxış bucaqları var.Bundan öncəki bəhslərdə az-çox if else operatorlarından istifadə etdik
if operatoruna bir daha nəzər salaq.

```
>>> i=4
>>> if i>0:
        print(True)


True
>>> i=4
>>> if i<9:
        print(False)


False
>>>
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
print("""\nTəqdim olunan şərti ad və soyad:
ad :Nigar
soyad:Məhərrəmova
""")
i='Nigar'
x='Məhərrəmova'
ad=input('Zəhmət olmasa adınızı yazın:')
soyad=input('soyadınızı yazın:')
if ad==i or soyad==x:
    print('adınız:{}'.format(i))
    print('soyadınız:{}'.format(x))
```

```
Təqdim olunan şərti ad və soyad:
ad :Nigar
soyad:Məhərrəmova

Zəhmət olmasa adınızı yazın:Nigar
soyadınızı yazın:Məhərrəmova
adınız:Nigar
soyadınız:Məhərrəmova
>>>
```

Ilk əvvəl şərti ad və soyad qeyd etdik daha sonra istifadəçidən bu ad,soyadı girməsini tələb etdik.Daha sonra if operatorundan istifadə edərək bu ad,soyadı ekrana çap etdik

Qarşılaşdırma işarələri

İşarə	Mənası
>	Böyükdür
<	Kiçikdir
>=	Böyük bərabərdir
<=	Kiçik bərabərdir

==	Bərabərlik
!=	Bərabər deyil

İlk kodlarımızda istifadə etdiyimiz qarşılaşdırma işarələrini bir cədvələ tərtib edərək bizə rahat olmasını təmin etdik.Böyük və kiçikdir işarələrindən istifadə etdik, indidə digərlərinə baxaq.

```
>>> if a>=19:
        print(True)


True



>>> a=19
>>> if a<=19:
        print(False)


False

>>> a=19
>>> if a==19:
        print(True)


True

 >>> a=19
 >>> if a==19:
         print(True)


 True



 >>> a=19
 >>> if a!=19:
         print(False)
```

False

```
>>> i=20
>>> i==19
False
>>> i>=19
True
>>> i<=19
False
>>> i!=19
True
>>> i>19
True
>>> i<19
False
>>>
```

Yuxarıda istifadə etdiyimiz True,False parametrləri şərti parametrlərdir.True-təsdiq   False-inkar ı bildirir.cavabın təsdiq ya inkarda olduğunu biz özümüz şərti parametrlərlə qeyd etdik.Yəni biz hətda 2+2=4 ifadəsini bu parametrlər vasitəsilə yanlış olduğunu argument olaraq yaza bilərik.Amma son i=20 ifadədən başlayan True ,False parametrlərini python bizə söylədi.Biz sadəcə i=20 ifadəsini qarşılıqlı digər rəqəmlərlə test etdik.

```
>>> i=2
>>> x=2
>>> cavab=i+x
>>> if cavab:
	print(cavab,True)
4 True
>>> if cavab:
	print(cavab,False)
4 False
>>>
```

Gördüyümüz kimi ilk 2+2=4 ifadəsini True olaraq biz qeyd etdik.Riyazi cavaba görədə bu belədir.Amma sonra cavabın 4 olduğunu inkar etdik.Bunu sadəcə biz qeyd etməyimizlə çap etdirdik.Python bunu görməzdən gəldi.Bu tip yanaşmalar qeyri səlis məntiq nəzəriyyəsinə

aiddir.Bunun ingiliscəsi fuzzy logic adlanır.pythonda bununla bağlı paketlər də var.Maraqlanmaq istəsəniz aşağıdakı ünvana nəzər yetirin.
https://pypi.python.org/pypi/scikit-fuzzy

Mövzumuza davam edərək if operatoru ilə daha geniş kod blokları yaza bilmədiyimiz üçün dərhal else operatoruna keçək.Bərabər işlənən bu cütlük daha prespektiv kodlara imza ata bilirlər.

else operatoru

bu operator if operatoru ilə bərabər işlənən operatordur.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
parol='digikam657'
passw=input('write you passw:')
if passw==parol:
    print('Okey,{} password is right'.format(passw))
else:
    print('{} password is wrong'.format(passw))
```

```
write you passw:digi
digi password is wrong
>>>
write you passw:digikam657
Okey,digikam657 password is right
```

Yuxarıdakı kodlarımızda şərti olaraq digikam657 parolunu qeyd etdik,daha sonra istifadəçidən parolu girməsini tələb etdik.if operatoru ilə istifadəçinin yazdığı parolu şərti parol ilə qarşılaşdırdıq.

Əgər   parol   bərabərdisə   parol a

if   passw   ==   parol :

Daha sonra else operatoru ilə əgər qarşılaşdırma yanlışdırsa ekrana
else:
    print('{} password is wrong'.format(passw)) çap etsin kodunu yazdıq.

elif operatoru
elif operatoru davamlı olaraq ifadələri qarşılaşdırmağa kömək edir.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
parol='digikam657'
i='aventure12'
passw=input('write you passw:')
if passw==parol:
    print('Okey,{} password is right'.format(passw))
elif passw==i:
    print('Okey,{} password is right'.format(i))
else:
    print('{} password is wrong'.format(passw))
```

```
write you passw:aventure12
Okey,aventure12 password is right
>>>
```

Riyazi operatorlar

Bu operatorları bir cədvələ yerləşdirək.

+	Toplama
-	Çıxma
/	Bölmə
*	Vurma
**	Qüvvət
%	Qalıq ifadə(modul)

```
>>> 12+9
21
>>> 23+34
57
>>> i=12
>>> a=45
>>> i+a
57
>>>
```

Qeyd edimki buradakı toplama işarəsi (+) öz funksiyasını riyazi olaraq yerinə yetirir.Amma string cinslərində toplama işarəsindən istifadə etsək,tamamilə başqa nəticə alarıq.

```
>>> 'Jonny'+' Dep'
'Jonny Dep'
>>> 'www.hackerteam'+'.'+'com'
'www.hackerteam.com'
>>>
>>> '*'+'.'+'/'+'#'
'*./#'
>>>
```

Ifadələrimizdə düşünürəm anlaşılmayan məsələ yoxdur.ifadələri dırnaq içinə salaraq toplama işarəsi vasitəsilə bir-birinə topladıq.

```
>>> 34-12
22
>>> 45-34
11
>>> i=16
>>> x=56
>>> i-x
-40
>>> x-i
40
>>>
```

```
>>> i=16
>>> x=56
>>> i/x
0.2857142857142857
>>> x/i
3.5
>>> 23/56
0.4107142857142857
>>> 23/12
1.9166666666666667
>>> 12/4
3.0
>>>
```

```
>>> 2*2
4
>>> 4*3
12
>>> i=12
>>> x=3
>>> i*x
36
```

```
>>>
```

vurma operatorundan tək riyazi hesablamalarda istifadə olunmur.Necə toplama işarəsi string cinslərdə istifadə olunduğu kimi,vurma operatoru da işarələrin hasilində istifadə olunur.
Misallara baxaq

```
>>> '*'*12
'************'

>>> '-'*34
'----------------------------------'
>>>
```

Bu metod vasitəsilə yazacağınız proqram əvvəlini görünüşlü edə bilrsiniz

Növbəti istifadə edəcəyimiz operator qüvvət operatorudur(**)

```
>>> 2**2
4
>>> 3**2
9
>>> i**2
144
>>> a**3
91125
>>> t**4
Traceback (most recent call last):
  File "<pyshell#38>", line 1, in <module>
    t**4
NameError: name 't' is not defined
>>> q**2
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    q**2
NameError: name 'q' is not defined
>>>
```

Yuxarıdakı xətalardan aydın oldurki qüvvət yalnız ədədlərlə aparılan əməliyyatdır(int() float() cinsləri)

```
>>> 3.2**2
10.240000000000002
>>>
```

qalıq ifadə(modul) modul operatoru %

Bu operator sayını digərinə böldükdə qalan qalığı ekrana çap etməyə yardımçı olur.(%)

```
>>> 3%1
0               3-ü 1-ə böldükdə qalan qalıq 0-dır
>>> 34%12
10              34-ü 12-ə böldükdə qalan qalıq 10-dur
>>> 23%12
11              23-ü 12-ə böldükdə qalan qalıq 11-dir
>>> 25%12
1               25-i  12-ə böldükdə qalan qalıq 1-dir
>>>
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
reqem=int(input('bir rəqəm yazın:'))
if reqem%2==0:
    print('yazdığınız rəqəm, {} cüt ədəddir'.format(reqem))
else:
    print('yazdığınız rəqəm, {} tək ədəddir'.format(reqem))
```

```
bir rəqəm yazın:25
yazdığınız rəqəm, 25 tək ədəddir
>>>

bir rəqəm yazın:3
yazdığınız rəqəm, 3 tək ədəddir
```

```
>>>

bir rəqəm yazın:34
yazdığınız rəqəm, 34 cüt ədəddir
>>>
```

if reqem%2==0 ifadəmizdə istifadəçinin yazdığı rəqəm 2-yə bölündükdə qalan qalıq 0 olarsa cavab cüt,yox 0-deyilsə cavab tək olaraq çap edilsin.

bool () operatoru

Necə True və False ifadələri təsdiq inkar olduğu kimi bool operatoru da iki qiymət alır.1 və 0

1-True
0-False

bool operatoru istənilən string argumentinə True ifadəsini verir.Amma saylara gəlincə 1 və 0 dan başqa eləcədə boşluq, bizim qarşılaşdırmayacağımız ədədləri də false olaraq ifadə edir.

```
>>> i=1
>>> i==1
True
>>> bool(i)
True
>>> bool('')
False
>>> bool(' ')
True
>>> bool(1)
True
>>> bool(0)
False
>>> bool(12)
True
```

```
>>> bool('techaz')
True
>>>
```

Nəticə etibarı ilə 0 və boşluq ifadələrindən başqa istənilən ifadə True-yəni təsdiqdir.

Digər riyazi operator hesablama üsulları

+=	a=12 a+=4 və ya 12+4
-=	a=12 a-=4 və ya 12-4
/=	a=12 a/=4 və ya 12/4
*=	a=12 a*=4 və ya 12*4
%=	a=12 a%=4 və ya 12%4
**=	a=12 a**=4 və ya 12**4
//=	a=12 a//=4 və ya 12//4

```
>>> a=12
>>> a+=4
>>> print(a)
16
>>> a=12
>>> a-=4
>>> print(a)
8
>>> a=12
>>> a/=4
>>> print(a)
```

```
3.0
>>> a=12
>>> a*=4
>>> print(a)
48
>>> a=12
>>> a%=4
>>> print(a)
0
>>> a=12
>>> a**=4
>>> print(a)
20736
>>> a=4
>>> a=12
>>> a//=4
>>> print(a)
3
>>>
```

round() funksiyası

round() funksiyası kəsirli sayların yuvarlaqlaşdırılmış nəticəsini göstərir.

```
>>> i=23.5
>>> round(i)
24
>>> i=23.4
>>> round(i)
23
>>>
```

İlk ifadəmizdə nəticə etibarı ilə 23.5-ə 24-aldıq.Əgər say 5-kəsrinə bərabər və ya keçmiş olarsa say yuvarlaqlaşdırılır.İkinci ifadəmizdə 5 və 5-dən kiçik(23.4) olduğu üçün say yuvarlaqlaşdırılmadı.

funksiya iki argument ala bilir.İlk argument yuvarlaşdırılacaq kəsirli say,digəri isə kəsirdən sonrakı ədədlər

```
>>> i=24.7
```

```
>>> round(i,1)
24.7
>>> round(i,2)
24.7
>>> i=12.567
>>> round(i,2)
12.57
>>> round(i,3)
12.567
>>>
```

funksiyanı qavramaq üçün bir neçə misallar yazaraq nə kimi nəticə verdiyini başa düşməyə çalışın.

Əvvəlki bəhslərimizdə qüvvət işarəsi olaraq iki ulduz parametrindən istifadə edirdik.Amma bu parametri əvəzləyəcək pow() funksiyası var.Misallara baxaq

```
>>> pow(i,2)
157.92948900000002
>>> pow(3,2)
9
>>>
```

Bəli 3**2 =9 edir

pow() funksiyası 3 argument ala bilir

```
>>> pow(3,2,2)
1
>>> pow(2,2,1)
0
>>>
```

Yuxarıdakı ifadə,2**2=4 və 4%1 nəticə 0.
üçüncü argument modul bölmə əməliyyatını əvəz edir.Sözlə desək,2-ni 2 qüvvətinə yüksəlt və cavabı 1-ə böl sonda qalığı bizə göstər.bəli 4-sayı 1-ə böldükdə heç bir qalığı qalmır.Ondan öncəki ifadədə isə 3-ü 2-qüvvətinə yüksəlt cavabı 2-yə böl.Bəli 9 2-yə böldükdə qalıq 1 qalacaq. 3**2%2=1

enumerate()   funksiyası

funksiya verilənlərin daxilindəki argumentlərin hər birini bir rəqəmə ataraq sıralayır.Misallardan daha aydın olacaq.

```
>>> print(enumerate('python'))
<enumerate object at 0x7fee039e6c18>
>>> print(*enumerate('python'))
(0, 'p') (1, 'y') (2, 't') (3, 'h') (4, 'o') (5, 'n')
>>>
```

və ya

```
>>> for i in enumerate('python'):
        print(*i)



0 p
1 y
2 t
3 h
4 o
5 n
>>>

>>> for i in enumerate('python'):
        print(i)



(0, 'p')
(1, 'y')
(2, 't')
(3, 'h')
(4, 'o')
(5, 'n')
>>>
```

abs() funksiyası

Funksiya sayın mütləq dəyərini göstərir.

```
>>> abs(-4)
4
>>> abs(12.3)
12.3
>>> abs(-12.3)
12.3
>>>
```

max,min () funksiyaları

Parametrlər ifadənin daxilində maksimum və minimum dəyərləri göstərir.Misallara baxaq

```
>>> i='1,2,3,4,5,6'
>>> max(i)
'6'
>>> max([i])
'1,2,3,4,5,6'
>>> for i in max(i):
	print (i)


6
>>> i=2,1,4,6,48,7,5
>>> max(i)
48
>>> min(i)
1
>>>
```

key argumenti

max və min () funksiyaları string cinslərində xəta verir

```
>>> for x in len(i):
	print(max(x))
```

```
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    for x in len(i):
TypeError: 'int' object is not iterable
>>>
```

str cins ifadələrində uzunluğu böyük və kiçik olan dəyərləri təyin etmək üçün key argumentindən istifadə edəcik.

```
>>> i='Azerbaijan','Island','Dublin','England'
>>> print(max(i,key=len))
Azerbaijan
>>>
```

eyni qayda ilə

```
>>> print(min(i,key=len))
Island
```

funksiyalar (max(),min())soldan-sağa doğru tamamilə ifadəni oxuyub,başdan ən böyük və ən kiçik dəyərləri çap edir.Yəni ifadə daxilində bir neçə eyni uzunluqda olan dəyərləri çap etməyə qadir deyil.

sum() funksiyası

sum-summa yəni cəmi.İfadə daxilində sayları toplayaraq yekun nəticəni bizə göstərir.İki ədəd argument ala bilir.

```
>>> i=1,2,3,4,5
>>> sum(i)
15
```

Toplananların üzərinə digər sayı əlavə etmək üçün

```
>>> i=1,2,3,4,5
>>> sum(i)
15
>>> sum(i,6)
```

```
21
>>>
```

sum(i,6) ifadəsindəki 6-rəqəmini i=1,2,3,4,5,6 kimi başa düşün.

divmod() funksiyası

Funksiya, ədədi ədədə böldükdə cavabı və qalan qalığı göstərir.Bir işi modul(%) parametrinə bənzəyir,amma üstünlüyü cavabı da ekrana çap etməsidir.

```
>>> divmod(2**2,2)
(2, 0)
>>> divmod(10,2)
(5, 0)
>>> divmod(pow(3,2),2)
(4, 1)
>>>
```

bin() funksiyası,binar ədədlər

```
>>> bin(1)
'0b1'
>>> bin(10)
'0b1010'
>>> int('0b1010',2)
10
>>> bin(50)
'0b110010'
>>> int('0b110010',2)
50
>>> a=int('01100000', 2)
>>> b=int('00100110', 2)
>>> bin(a & b)
'0b100000'
>>> bin(a | b)
'0b1100110'
>>> bin(a ^ b)
```

'0b1000110'

and or, not operatorları

and
Dilimizə ‘və’ kimi tərcümə olunur.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
ad=input('write you name:')
soyad=input('write you surname:')
if ad=='Python' and soyad=='programming':
    print('OK,this is right')
else:
    print('wrong!')
write you name:Python
write you surname:programming
OK,this is right

write you name:Elli
write you surname:vladi
wrong!
>>>
```

or operatoru
Dilimizə ‘və ya’ kimi tərcümə olunur

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
number=int(input('write number:'))
if number>100 or number>=100:
    print('OK,this is right')
elif number<100 or number<=100:
    print('wrong!')

write number:80
```

```
wrong!
>>>

write number:100
OK,this is right
>>>
```

not operatoru

Dilimiz ‘yox ‘ kimi tərcümə olunur.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
lst=['elder','wolf','summer','book']
string=input('write number:')
if string not in lst:
    print('wrong')
else:
    print('right'.format(string))
```

```
write number:wolf
right
>>>

write number:elder
right
>>>

write number:er
wrong
>>>
```

Kodlarımızda if string not in lst   ifadəsi vasitəsilə istifadəçidən aldığımız string ifadənin listimizdə olmadığına dair əmr verdik.Daha dəqiq desək

əgər	ifadə	yoxdur	içində	listin
if	string	not	in	lst

Əgər python2 dən xatırlayırsınızsa biz not in ifadəsini istifadə edə

biləcəyimiz modul metodlarına da baxa bilirdik.

```
>>> import re
>>> dir(re)
['A', 'ASCII', 'DEBUG', 'DOTALL', 'I', 'IGNORECASE', 'L', 'LOCALE', 'M',
'MULTILINE', 'S', 'Scanner', 'T', 'TEMPLATE', 'U', 'UNICODE', 'VERBOSE', 'X',
'_MAXCACHE', '__all__', '__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', '__version__',
'_alphanum_bytes', '_alphanum_str', '_cache', '_cache_repl', '_compile',
'_compile_repl', '_expand', '_locale', '_pattern_type', '_pickle', '_subx',
'compile', 'copyreg', 'error', 'escape', 'findall', 'finditer', 'fullmatch',
'match', 'purge', 'search', 'split', 'sre_compile', 'sre_parse', 'sub', 'subn',
'sys', 'template']
```

re modulunun içində olan __loader__ və s ifadələri ekrana çap etməmək üçün aşağıdakı yoldan istifadə edəcəyik.

```
>>>  for i in dir(re):
         if '_' not in i:
                print(i)
```

Və gördüyünüz kimi ekrana alt-tire olan metod və funksiyalar listələnmədi.

Kodlarımızda not ifadəsi ilə yanaşı in operatorunu da işlətdik.Çox zaman bu iki operator bərabər istifadə olunur.Bu sizin yazacağınız proqramlara bağlıdır.in operatoru həm digər operatorlarla bərabər,həm də ayrı ayrı istifadə oluna bilir.İndi də gəlin in operatoruna baxaq.

```
>>> i='underground'
>>> 'a' in i
False
>>> 'u' in i
True
>>> 'g' in i
True
>>> 'g' not in i
False
>>> 'a' not in i
```

```
True
>>>
```

Ilk başda i-hərfinə underground ifadəsi atdıq,daha sonra a-hərfinin i-nin içində olduğunu soruşduq,bizə False(inkar) ifadəsi verdi.Bəli doğrudur a-hərfi i-nin içində deyil.Digərləri də həmçinin.Amma son iki not operatoru ilə bərabər istifadə etdikdə

g	yoxdur	içində	i-nin
'g'	not	in	i

bizə False ifadəsi verdi.Yəni g-hərfi var.Sonuncu kodumuzda isə True aldıq(təsdiq).Bəli,yəni a- hərfi i-nin içində deyil.

misallar

```
>>> print('snapshot')
snapshot
>>> print("Elidor")
Elidor
>>> print("""digikam""")
digikam
>>> print(12)
12
>>> print('Alla','Natasha',23,'Nargiz')
Alla Natasha 23 Nargiz
>>> print('Alla','Natasha',23,'Nargiz',sep=',')
Alla,Natasha,23,Nargiz
>>>
>>> x='texno'
>>> 'i' in x
False
>>> 't' in x
True
>>> 'T' in x
```

```
False
>>>
```

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import division
print('''\nSalam,hesablayıcı programa xoş gəlmisiniz!''')
musbet='(1) toplama\n'
menfi='(2) cixma\n'
vurma='(3) vurma\n'
bolme='(4) bolme\n'
faiz='(5) faiz'
print (musbet,menfi,vurma,bolme,faiz)
sec=input("Yuxarıdakı əməllərdən birini seçin və enter'ə basın:")
if sec=='1':
    print ('siz toplama əməlini seçdiniz')
    a=int(input('ilk ədədi daxil edin:'))
    a1=int(input('ikinci ədədi daxil edin:'))
    print ('cavab=',a+a1)
elif sec=='2':
    print ('siz çıxma əməlini seçdiniz')
    a=int(input('ilk ədədi daxil edin:'))
    a1=int(input('ikinci ədədi daxil edin:'))
    print ('cavab=',a-a1)
elif sec=='3':
    print ('siz vurma əməlini seçdiniz')
    a=int(input('ilk ədədi daxil edin:'))
    a1=int(input('ikinci ədədi daxil edin:'))
    print ('cavab=',a*a1)
elif sec=='4':
    print ('siz bölmə əməlini seçdiniz')
    a=int(input('ilk ədədi daxil edin:'))
    a1=int(input('ikinci ədədi daxil edin:'))
    print ('cavab=',a/a1)
elif sec=='5':
    print ('siz faiz əməlini seçdiniz')
    a=int(input('ilk ədədi daxil edin:'))
    a1=int(input('ikinci ədədi daxil edin:'))
    print ('cavab=',a*a1/100)
```

```
else:
    print('wrong')
```

```
Salam,hesablayıcı programa xoş gəlmisiniz!
(1) toplama
 (2) cixma
 (3) vurma
 (4) bolme
 (5) faiz
Yuxarıdakı əməllərdən birini seçin və enter'ə basın:1
siz toplama əməlini seçdiniz
ilk ədədi daxil edin:2
ikinci ədədi daxil edin:4
cavab= 6
>>>
```

```
Salam,hesablayıcı programa xoş gəlmisiniz!
(1) toplama
 (2) cixma
 (3) vurma
 (4) bolme
 (5) faiz
Yuxarıdakı əməllərdən birini seçin və enter'ə basın:4
siz bölmə əməlini seçdiniz
ilk ədədi daxil edin:23
ikinci ədədi daxil edin:45
cavab= 0.5111111111111111
>>>
```

Hesablama proqramının yazılışında əgər səhviniz olarsa aşağıdakı ünvana daxil olub daha dəqiq görə bilərsiniz
http://pastebin.ubuntu.com/23169562/

Suallar

1. Aşağıdakılardan hansı doğrudur

a) a=’techaz’
    if ‘a’ in a

b) a=’techaz’
    if ‘a’ in a:

c) a=’techaz’
    if a in a:

d) a=’techaz’
    if ‘a’ in a;

2.Aşağıdakılardan hansı doğrudur

a) for i in a:
    print a

b) for i in a:
    print(i)

c) for i in a:
    print(a)

d) for i in a
    print(i)

3.Aşağıdakı ifadələrdən hansı yanlışdır.

a) print ‘Algida’,’Epsilon’,sep=’*’
b) print(‘Algida’,’Epsilon’,sep=’,’)
c) print(‘Algida’,’Epsilon’,end=’.\’)
d) print(‘Algida’,’Epsilon’,sep=’i’)

4) 90<=x<100 hansı aralıq doğrudur?

a) x=70 True   b) x=90 False   c) x=90 True   d)x=100 True

5. fayl=open('techaz.txt','w') aşağıdakılardan hansı doğrudur.

a) print('texas',sep='\n',file=fayl,flush=True)
b) print 'texas',sep='\n',file=fayl,flush=True’
c) print('texas',sep='\n',file=fayl,flush=False)
d) print(texas,sep='\n',file=fayl,flush=True)

id() funksiyası

Pythonda bu parametr ifadələrin passport nömrəsini göstərir.İfadənin yaddaşda tutduğu say nömrəsini çap edir.

```
>>> i=100
>>> id(i)
10927488
>>> b=100
>>> id(b)
10927488
>>> b==i
True
>>>
```

Gördüyümüz kimi a=100 və b=100 ifadələrinin id dəyəri eynidir(10927488) python ilk əvvəl a və b üçün yaddaşda yer ayırır və ardından id dəyərlərini qiymətləndirir.qarşılaşdırma yolu ilə (a==b) yoxladıqda bizə True cavabını verdi.İndidə gəlin bu ifadələrə başqa dəyər verərək id nömrələrini sorğuya çəkək.

```
>>> i=1002
>>> id(i)
139746279051568
>>> b=1002
>>> id(b)
139746279051600
>>> i==b
True
>>> id(i)==id(b)
False
>>>
```

Yuxarıda i və b hərflərinin dəyərlərini dəyişərək 1002 yazdıq.Və ardından bu ifadələrin id nömrəsini soruşduq və nəticədə fərqli qiymətlər aldıq.Əsas məsələ isə qarşılaşdırma işarəsi olan iki bərabərlik(==) i və b hərfləri üçün True verməsidir.Burdan da agah olurki bu iki bərabərlik i və b ifadələrinin yalnız dəyər qiymətlərinə(1002) baxır,id nömrəsinə deyil.Daha sonra biz id(i)==id(b) soruşduqda isə bizə False verdi.Bəli bu iki ifadənin id nömrələri fərqlidir.
Deməli ayrı ayrılıqda hər bir ifadənin fərqli id-nömrələri yəni passport nömrələri var.

is funksiyası

Dilimizə dır,dir,dur,dür kimi tərcümə olunur.İfadələri qarşılaşdırma metodudur.Bu operatoru tam anlamaq üçün gəlin misallara baxaq.

```
>>> a=1000
>>> a is id(1000)
False

>>> a='algebra'
>>> a is 'algebra'
True
>>>
```

while operatoru(loop)

Bu operator dilimizə ‘isə’ kimi tərcümə olunur.Operator qramatikamızda bağlayıcıdır .Bu ingilis dilinin qramatikasında da belədir.Bizə isə məhz pythonda hansı mənanı kəsb etməsidir.Pythonda da bu operator ya

True(təsdiq) yada False(inkar) parametrlərindən birini seçir,eləcədə qarşılaşdırma parametrləri ilə bərabər olaraq istifadə etmək olur.

 isə   doğru
 | |
while True: ifadə- doğru isə və ya doğru olduğu müddətcə

 isə   yanlış
 | |
while False: ifadə- yanlış isə və ya yanlış olduğu müddətcə

və ya

```
i=20
while i==1:
       print('Hello python')
```

Bu operator həmdə sirkulyasiya(dövr etmə) metodunu yerinə yetirir.Verdiyiniz dəyəri sona qədər nəzərdən keçirir və yenə başa dönür

```
>>> a=2
>>> while a<20:
              print('aa')
```

while sonsuz dövr etdiyi üçün yuxarıdakı kodlarımızın nəzəriyyəsi

a(2) 20-dən kiçik olduğu müddətcə ekrana (aa) çap et.

Və biz ctrl+c basmayınca sonsuz dəyəri (aa) ekrana çap edəcək

Biz bunu limitləndirərək sonsuzluğu sonlu dəyərə çevirə bilərik.

```
>>> a=2
>>> while a<20:
	a=a+1
	print(a)

3
4
5
6
....
....
19
20
>>>
```

Yuxarıdakı kodlarımızda ilk a-ya 2 sayını verdik,daha sonra a 20-dən kiçik olduğu müddətcə a-nın üzərinə bir gələrək(yəni başlanğıcı 3-dən olacaq,çünki a-nın ilk qiyməti ikidir.(2+1) bu dəyərləri çap et əmrini verdik.

Misallarda yazdığımız hesablama proqramına bir də qayıdaraq while operatoru ilə bərabər yazaq.

İlk əvvəl qrafik görünüşünü təşkil edək

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import division
print('''\nSalam,hesablayıcı programa xoş gəlmisiniz!''')
musbet='(1) toplama\n'
menfi='(2) cixma\n'
vurma='(3) vurma\n'
bolme='(4) bolme\n'
```

```
faiz='(5) faiz'
print (musbet,menfi,vurma,bolme,faiz)


while True:
    sec=input("Yuxarıdakı əməllərdən birini seçin və enter'ə basın:")
    if sec=='1':
        print ('siz toplama əməlini seçdiniz')
        a=float(input('ilk ədədi daxil edin:'))
        a1=float(input('ikinci ədədi daxil edin:'))
        print ('cavab=',a+a1)
    elif sec=='2':
        print ('siz çıxma əməlini seçdiniz')
        a=float(input('ilk ədədi daxil edin:'))
        a1=float(input('ikinci ədədi daxil edin:'))
        print ('cavab=',a-a1)
    elif sec=='3':
        print ('siz vurma əməlini seçdiniz')
        a=float(input('ilk ədədi daxil edin:'))
        a1=float(input('ikinci ədədi daxil edin:'))
        print ('cavab=',a*a1)
    elif sec=='4':
        print ('siz bölmə əməlini seçdiniz')
        a=float(input('ilk ədədi daxil edin:'))
        a1=float(input('ikinci ədədi daxil edin:'))
        print ('cavab=',a/a1)
    elif sec=='5':
        print ('siz faiz əməlini seçdiniz')
        a=float(input('ilk ədədi daxil edin:'))
        a1=float(input('ikinci ədədi daxil edin:'))
        print ('cavab=',a*a1/100)
    else:
        print('wrong')
```

Yuxarıda gördüyünüz float cins dəyişdirməsini yazmaqda məqsədimiz istifadəçi hər halda kəsirli say daxil edərsə problemlə qarşılaşmasın.Siz istəsəniz int -ilə əvəz edərək daha sonra kəsirli sayılar verin və nə kimi xətalarla üzləşəcəyinizi nəzərdən keçirin.

While operatorunu hara yerləşdirəcəyinizə diqqət edin həmişə.Çünki bu

operator kodları sona qədər oxuyub yenidən başa dönür.Aşağıdakı misala baxaq

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
ad=input('bir ad yazın:')
while True:
    if len(ad)<8:
        print('adınız səkkiz saydan azdır!')
    else:
        print('adınız qeydə alındı!')
```

Kodlarımızda while True: ifadəsini   ad=input('bir ad yazın:') ifadəsindən sonra yazdıq,yəni while operatoru tam else operatoruna qədər kodları oxuyub sonra if len(ad)<8: ifadəsinə geri dönəcək.Kodlarımızı çalışdırdıqda nə ilə nəticələndiyini gördük.Əgər 8-sayından az bir ad yazarsaq ekranda dayanmadan-davamlı olaraq   adınız səkkiz saydan azdır!   Ifadəsini görəcəyik,yox əgər 8-saydan çox yazarsaq davamlı olaraq   adınız qeydə alındı!   Ifadəsini görəcəyik.O zamana qədərki biz məcburi ctrl+c düymələrini basmasaq.

for operatoru

Dilimizə ‘üçün’ kimi tərcümə olunur.operatora dair misallara baxaq

```
>>> import os
>>> dir(os)
.............................
>>> for i in dir(os):
	if '_' not in i:
		print(i)
```

Yuxarıdakı kodlarla öncəki bəhslərdən tanışıq.

```
>>> a='python','java','c++'
```

```
>>> for i in a:
	print(i)


python
java
c++
>>> for i in a:
	print(i,'\tproqramlama dili')


python	proqramlama dili
java		proqramlama dili
c++	proqramlama dili
>>>

>>> a='python','java','c++'
>>> b='programming','application','arduino'
>>> for i in a:
	print(i,'\t{}'.format(b))


python	('programming', 'application', 'arduino')
java		('programming', 'application', 'arduino')
c++	('programming', 'application', 'arduino')
>>>
```

Əgər \t xüsusi işarəsini yazmasaq,ekran çapı bərbad görünəcək.

```
>>> for i in a:
	print(i,'{}'.format(b))

python ('programming', 'application', 'arduino')
java ('programming', 'application', 'arduino')
c++ ('programming', 'application', 'arduino')
>>>


>>> i=1234
```

```
>>> for x in i:
	print (x)



Traceback (most recent call last):
  File "<pyshell#192>", line 1, in <module>
    for x in i:
TypeError: 'int' object is not iterable
```

və xəta aldıq.Deməli string cinsində göstərilməyən ifadələri for operatoru oxuya bilmir

```
>>> x='12345'
>>> for i in x:
	print(i)


1
2
3
4
5
>>>
```

range() funksiyası

Dilimizə aralıq kimi tərcümə olunur.Ala bildiyi argumentlər string və integer cinsləridir.

range() funksiyası vasitəsilə(for operatoru ilə) ədədləri ekrana tökə bilərsiniz.

```
>>> for i in range(8):
	print (i)
```

```
0
1
2
3
4
5
6
7
>>>
```

Gördüyümüz kimi 0-dan 8-ə qədər olan ədədləri ekrana tökdü.
Əgər aralıq olaraq ifadələri bildirsək

```
>>> for i in range(0,8):
        print (i)


0
1
2
3
4
5
6
7
>>>
```

Ilk ifadə ilə sonuncu ifadəmiz eyni metodla ekrana ədədləri çap etdi.Yəni range(8) və range(0,8) ifadələri eynilik təşkil edir.

range() funksiyasından istifadə edərək bir proqram yazaq.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
passw=input('please write you passw:')
for i in passw:
    if len(passw) in range(0,8):
        print('Ok,you passw {}'.format(passw))
```

```
    else:
        print('you passw {} very long'.format(passw))
```

```
please write you passw:armadia
Ok,you passw armadia
Ok,you passw armadia
Ok,you passw armadia
Ok,you passw armadia
Ok,you passw armadia
Ok,you passw armadia
Ok,you passw armadia
>>>
```

Yuxarıdakı kodlarımızda len() funksiyasından istifadə edərək parolmuzun uzunluğunu range(0,8) aralığından çox olmamaq şərtilə istifadəçidən aldıq.Amma ekran çapında bir natamamlıq var.Çünki ekrana bir neçə dəfə Ok,you passw armadia ifadəsini çap etdi.Bunun qarşısını almaq üçün python bizə break operatorunu təklif edir.break operatoru haqqında irəli bəhslərimizdə geniş danışacıq.

İndidə gəlin break operatorun əlavə edərək kodlarımıza biraz görünüş əlavə edək.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
passw=input('please write you passw:')
for i in passw:
    if len(passw) in range(0,8):
        print('Ok,you passw {}'.format(passw))
        break
    else:
        print('you passw {} very long'.format(passw))
        break
```

```
please write you passw:armagia
Ok,you passw armagia
>>>
```

```
please write you passw:galsjgljgsag
you passw galsjgljgsag very long
>>>
```

Və ekran çapından gördüyümüz kimi ifadə yalnız bir dəfə çap olur.

range() funksiyası daxilində üç argument də ala bilir.

```
>>> for i in range(0,10,3):
	print(i)


0
3
6
9
>>>
```

range(0,10,3) ifadə,0 və 10 arlığındakı rəqəmlərə 0 daxil olmaqla üzərinə 3 gələrək çap et.

Geriyə sıralamada isə mənfi işarəsindən istifadə edəcəyik

```
>>> for i in range(8,0,-1):
	print(i)


8
7
6
5
4
3
2
1
```

```
>>>
```

və ya

```
>>> for i in range(8,0,-4):
	print(i)


8
4
>>>
```

ekran çapından göründüyü kimi for operatoru ilə istənilən ifadələr y-oxu üzrə çap olunur.Digər metodlardan istifadə edərək range funksiyasının aldığı argumentləri çap edək.

```
>>> print(range(0,10))
range(0, 10)
>>> print(range(6))
range(0, 6)
```

kodlarımızda gördüyünüz kimi ifadələr,range() funksiyasının daxilində argumentləri necə yazmışıqsa eləcədə çap olunur.

```
>>> print(*range(6))
0 1 2 3 4 5
>>>
```

Yuxarıdakı metodumuzda isə hasil işarəsi yazaraq 6-ya qədər olan sayları ekranda göstərə bildik.
İndidə gəlin sep metodundan istifadə edərək bunu gerçəkləşdirək.

```
>>> print(*range(8),sep='.')
0.1.2.3.4.5.6.7
```

```
>>>
```

sep metodu vasitəsilə saylarımızın arasına nöqtə işarəsini əlavə edə bildik.

```
>>> print(*range(8),end='./')
0 1 2 3 4 5 6 7./
>>> print(*range(8),sep='\t')
0	1	2	3	4	5	6	7
>>>
```

len() funksiyası

len() funksiyası uzunluq anlayışını ifadə edir.Bu funksiya əsasən string cinslərdə baş tutur,amma saylarla bunu reallaşdırmaq istəsək cins dəyişdirmə tiplərindən istifadə etməliyik.

```
>>> len('Europia')
7
>>> a='karavan'
>>> len(a)
7
>>> len(123456)
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    len(123456)
TypeError: object of type 'int' has no len()
>>>
```

Sonuncu ifadəmizdə xəta aldıq.xətada deyilirki int tipi len funksiyası üçün deyil.(TypeError).
Bunu

```
>>> len(str(123456))
6
>>>
```

yazaraq uzunluğunu öyrənə bilərik.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    parol=input('lütfən parolunuzu yazın:')
    if len(parol)<8:
        print('zəhmət olmasa səkkiz saydan az parol yazmayın')
    else:
        print('parolunuz {}'.format(parol))
```

```
lütfən parolunuzu yazın:algei
zəhmət olmasa səkkiz saydan az parol yazmayın
lütfən parolunuzu yazın:germany1234
parolunuz germany1234
lütfən parolunuzu yazın:
```

pass,break,continue operatorları

pass operatoru

pass operatoru ingiliscədən ötürmək,keçmək kimi tərcümə olunur.Pythonda da bu operator əvvəlki kod blokunu passivləşdirərək,növbəti kod blokunun aktivləşməsini həyata keçirir.Daha açıq desək bu ifadə görməzdən gəlir.Misallarla operatorun funksiyası aşkar olacaq.Misallara baxaq.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    parol=input('lütfən parolunuzu yazın:')
    if len(parol)<8:
        pass
    else:
```

```
        print('parolunuz {}'.format(parol))
```

```
lütfən parolunuzu yazın:
lütfən parolunuzu yazın:34
lütfən parolunuzu yazın:45
lütfən parolunuzu yazın:
lütfən parolunuzu yazın:
```

Biz yuxarıdakı kodlarımızda if len(parol)<8: yazmaqla,yəni istifadəçi əgər 8-sayıdan az ifadə daxil edərsə ötür keç növbəti kodlara.Bu ötürmə əməliyyatını pass operatoru ilə gerçəkləşdirdik.

```
if len(parol)<8:
        pass
```

Yəni istifadəçinin girdiyi az sayda parol və ya boşluq buraxıb enter-ə basarsa python heç bir söz söyləmədən else operatoruna baxacaq. Ekran çapında göründüyü kimi ilk biz boş buraxaraq enter-ə basdıq və heç bir ifadə çap olunmadı ikincisində də həmçinin 34 yazmaqla 8-saydan az olduğu üçün yenə pass operatoru bunu ötürdü və sairə belə ardıcıl olaraq davam edə bilərik.

break opertoru

Dilimizə kəsmək,sındırmaq kimi tərcümə olunur.Pythonda da mənası qədər iş görür.Bu operator vasitəsilə davam edən kod bloklarını yarıda kəsə bilərik.Misallara nəzər yetirək.

```
#!/usr/bin/env python
```

```
# -*- coding: utf-8 -*-
while True:
    ad=input('bir ad yazın:')
    if len(ad)<8:
        print('adınız səkkiz saydan azdır!')
        break
    else:
        print('adınız qeydə alındı!')
```

```
bir ad yazın:niko
adınız səkkiz saydan azdır!
>>>
```

Kodlarımızda gördüyünüz kimi break operatorunu əlavə edərək istifadəçi əgər 8-saydan az ifadə daxil edərsə proqramı yarıda kəsəcik.

Və qeyd edimki əgər siz dövrü kodlardan kənarda break operatorundan istifadə etsəniz xəta alacaqsınız

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
ad=input('bir ad yazın:')
if len(ad)<8:
    print('adınız səkkiz saydan azdır!')
else:
    print('adınız qeydə alındı!')
    break
```

```
SyntaxError: 'break' outside loop
```

Və xəta aldıq.loop ifadəsi dövrü operatoru ifadə edir.Yəni Bu ya for yada while ola bilər.Bu iki operatordan kənarda break ifadəsi istifadə olunmur.

continue operatoru

Bu operator dilimizə ‘davam,davamı’ kimi tərcümə olunur.Tərcüməsi qədər də python da iş görür.Biz break operatorundan kodlarımızda istifadə edərkən kodlarımızı yarıda kəsirdiksə bu operator tam tərs işi görərək kodları davam etməyə məcbur edir.Aşağıdakı misallara baxaq.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    ad=input('bir ad yazın:')
    if len(ad)<8:
        print('adınız səkkiz saydan azdır!')
        continue
    else:
        print('adınız qeydə alındı!')
        break
```

```
bir ad yazın:asd
adınız səkkiz saydan azdır!
bir ad yazın:er
adınız səkkiz saydan azdır!
bir ad yazın:effe
adınız səkkiz saydan azdır!
bir ad yazın:sad
adınız səkkiz saydan azdır!
bir ad yazın:
```

Yuxarıda kodlarımıza ‘continue’ əlavə edərək,istifadəçinin 8-saydan artıq ad yazacağına qədər proqramı davam etdirməyə məcbur etdik.O zamana qədərki istifadəçi 8-saydan artıq ifadə yazarsa proqram break operatoru vasitəsilə kəsiləcək.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    ad=input('bir ad yazın:')
```

```
    if len(ad)<8:
        print('adınız səkkiz saydan azdır!')
        continue
    else:
        print('adınız qeydə alındı!')
        break
```

```
bir ad yazın:asd
adınız səkkiz saydan azdır!
bir ad yazın:er
adınız səkkiz saydan azdır!
bir ad yazın:effe
adınız səkkiz saydan azdır!
bir ad yazın:sad
adınız səkkiz saydan azdır!
bir ad yazın:azerbaijan
adınız qeydə alındı!
>>>
```

Gördüyümüz kimi sonda 8-saydan yuxarı ifadə daxil etdik və proqram ‘adınız qeydə alındı!’ ifadəsini çap edib dayandı.

pythonda modul(%)

Riyazi hesablamalarda istifadə etdiyimiz bu parametr,hesablamalardan əlavə başqa funksiyanı icra edir.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
num = int(input('Now enter a number: '))
print ('Doubling your number:',num,'=', (int(num) * 2))
```

```
Now enter a number: 22
Doubling your number: 22 = 44
```

Yuxarıda gördüyünüz num ifadəsi istifadəçi tərəfindən yazılan saydır.Gəlin moduldan istifadə edərək istifadəçinin daxil etdiyi rəqəmi çap edək.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
num = int(input('Now enter a number: '))
print ('Doubling your number: %s' %num,(num * 2))

Now enter a number: 44
Doubling your number: 44 88
>>>
```

Kodlarımızda %s işarəsi gördüyünüz kimi dırnaq işarəsinin içində yazıldı.və daha sonra modulu tək yazaraq num ifadəsini dırnaq işarəsinin içinə saldıq.Əgər anlaşılmayan tərəfi varsa aşağıdakı misala nəzər yetirin

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
num = int(input('Now enter a number: '))
print ('Doubling %s your number:'%num,(num * 2))


Now enter a number: 22
Doubling 22 your number: 44
>>>
```

Yuxarıdakı kodlarımızda isə num ifadəsini dırnaq içində əvvələ yazdırdıq %s vasitəsilə.

format()   metodunun əvəzləyicisi kimi

% işarəsi

Parametr format metodunun işini yerinə yetirir.

```
>>> '{} and {} is very best program language'.format('Python','Java')
'Python and Java is very best program language'
>>>
```

format metodu ilə ifadəmizi yuxarıdakı kimi yazırdıqsa

%s parametri ilə

```
>>> '%s and %s is very best program language'%('Python','Java')
'Python and Java is very best program language'
>>>
```

Gördüyümüz kimi %s parametri də eyni işi gördü.
Niyə s-hərfi? Çünki string cinsləri üçün s-str olmasına rəğmən %s -parametri kimi ifadə olunur.Gəlin bir başqa hərf yazaraq necə nəticə çıxacağına baxaq

```
>>> '%d and %d is very best program language'%('Python','Java')

Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    '%d and %d is very best program language'%('Python','Java')
TypeError: %d format: a number is required, not str
>>>
```

və gördüyümüz kimi xəta aldıq.Çünki d-hərfi decimal saylar,eləcədə tam ədədlər(integer) üçün nəzərdə tutulub.
Gəlin Python və Java-nı integerlə əvəz edək

```
>>> '%d and %d is very best program language'%(12,34)
'12 and 34 is very best program language'
>>>
```

Və ekran görüntüsündə heç bir xəta almadıq.

```
>>> '%s and %s is very best program language'%(12,34)
'12 and 34 is very best program language'
>>>
```

Yuxarıdakı kodlarda isə integer cinsini string cinsinə çevirərək çap etdi

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
name=input('write you name:')
surn=input('write you surname:')
print('name %s and surname %s '%(name,surn))
```

```
write you name:Rashad
write you surname:Garayev
name Rashad and surname Garayev
>>>
```

format() metodunda biz cəm mötərizədə ifadə ardıcıllığı üçün say bildirirdiksə bu (%s) parametrində yoxdur.
Yəni

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
name=input('write you name:')
surn=input('write you surname:')
print('name {1} and surname {0} '.format(name,surn))
```

```
write you name:Rashad
write you surname:Garayev
name Garayev and surname Rashad
>>>
```

```
>>> for i in range(1,20):
	print('%s'%i)
```

%d və %i

Parametrləri decimal və integer saylarını çap etmək üçün istifadə olunur.Misallara baxaq

```
>>> print('%d və %d ədədləri'%(12,16))
12 və 16 ədədləri
>>> print('%i və %i ədədləri'%(12,16))
12 və 16 ədədləri
>>>
```

Eləcədə parametlərin içində(%6d və ya %6i) məsafə-sayını yazmaqla məsafə göstərə bilərsiniz

```
>>> print('%1d və %d ədədləri'%(12,16))
12 və 16 ədədləri
>>> print('%14d və %d ədədləri'%(12,16))
            12 və 16 ədədləri
```

%1d -yazmaqla sətir başından məsafə kənarlaşdı.Sonuncu ifadəmizdə isə %14d yazmaqla sətir başından məsafə saxladıq.

```
>>> print('%d və %d ədədləri'%(12.0,16.0))
12 və 16 ədədləri
```

float cinsində göstərdiyimiz ədədləri %d parametri çap etmədi.

Bunun üçün %f (yəni float) -işarəsi float cinslərdə istifadə olunur.

floating point decimal

```
>>> print('%f və %f ədədləri'%(12,16))
12.000000 və 16.000000 ədədləri
>>> print('%f və %f ədədləri'%(12.5,16.5))
12.500000 və 16.500000 ədədləri
>>>
```

floating point exponential

Bu ədədlərə e-ədədi yəni qüvvət kiçiltmələrini misal çəkmək olar.

```
>>> print('%e '%3.14e-10)
3.140000e-10
```

Xətalar. try , except blok operatorları

Biz hər zaman xətalarla qarşılaşırıq.Müxtəlif tipli xəta növləri var.Biz bir proqram yazaq və qrafik hissəsi gözəl görünüşə malik olsun.Ancaq istifadəçi proqramı çalışdırdıqda bir xəta ilə qarşılaşarsa,bu zaman bütün əməyiniz hədər olacaq.Kodları özümüz düşünüb və nəzəriyyəmizə əsaslanaraq ardıcıllığı tətbiq edirik.Amma kod yazmanın ən birinci qanunu xətaları qabaqlamaq və bunların həlli yollarıdır.Çünki istifadəçini istədiyimiz kimi bəzən yönləndirə bilmirik.Məsələn yazdığımız bir kodlarda istifadəçidən yalnız say girməsini tələb etmək üçün mütləq ilk başda zəhmət olmasa yalnız say girin yazası deyilik.Bu bərbad olar.kompyuter proqrama dilini labaratoriyada yazanlar bütün bunları anladığı üçün bizə bu xətaları qabaqla yolları üçün operatorlar təklif edirlər.Pythonda da xətaları qabaqlama metodları var.Gəlin aşağıdakı xəta növünə baxaq.

```
>>> a=int(input('write only number:'))
```

```
write only number:asd
Traceback (most recent call last):
  File "<pyshell#53>", line 1, in <module>
    a=int(input('write only number:'))
ValueError: invalid literal for int() with base 10: 'asd'
>>>
```

Gördüyünüz kimi biz istifadəçidən say,rəqəm tələb etsəkdə istifadəçi bizə qarşı hərf yazarq xəta aldı və proqram heç bir işə yaramadı.İndi də gəlin xətanı araşdıraq.

ValueError: invalid literal for int() with base 10: 'asd' xətada deyilirki ifadə yalnız integer cinsi ala bilər string-tipi deyil.Bu xətanın növünü təyin etmək üçün ən başda olan sətrə nəzər salın.ValueError-deməli xəta növü ValueError oldu.Bu xətanı qabaqlamaq üçün python bizə try və except xəta bloklarını təklif edir.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
     try:
         a=int(input('bir rəqəm yazın:'))
         print(a+4)
     except ValueError:
         print('zəhmət olmasa yalnız rəqəm girin!')
```

```
bir rəqəm yazın:asd
zəhmət olmasa yalnız rəqəm girin!
bir rəqəm yazın:1234
1238
bir rəqəm yazın:er
zəhmət olmasa yalnız rəqəm girin!
bir rəqəm yazın:
```

kodlarımızda a=int(input('bir rəqəm yazın:')) -ifadəsini try blokunun içi
print(a+4)
-nə salmaqda məqsədimiz,bu blokun içində hər hansı bir ValueError xətası olarsa except blokuna keçid et və except blokundakı ifadəni

göstərsin.

ZeroDivisionError xətası

Bu xəta növü adından göründüyü kimi zero-0 ilə bağlı xətadır.bu xəta əsasən 0-a bölmə əməliyyatı edərkən qarşımıza çıxır.Bu xəta ilə bağlı misala baxaq.

```
>>> 4/0
Traceback (most recent call last):
  File "<pyshell#55>", line 1, in <module>
    4/0
ZeroDivisionError: division by zero
>>>
```

Yeni xəta növümüz ZeroDivisionError -dur.Bu xəta növünü qabaqlamaq üçün yuxarıdakı kodlarımıza bəzi ifadələr əlavə edək

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
     try:
         num=int(input('bir rəqəm yazın:'))
         num1=int(input('bir rəqəm yazın:'))
         print(num/num1)
     except ValueError:
         print('zəhmət olmasa yalnız rəqəm girin!')
     except ZeroDivisionError:
         print('0-a bölmə əməli doğru deyil')
```

Və kodlarımıza except ZeroDivisionError: bloku əlavə etdik.Əgər istifadəçi ədədi 0-a bölərsə, print('0-a bölmə əməli doğru deyil') ifadəsi çap olunacaq.Bu xəta ilə yanaşı ilk yazdığımız xəta növü də qüvvədədir.Ardıcıllığın isə heç bir vacibliyi yoxdur.try operatoru sona qədər except -lə başlayan xəta növlərini nəzərdən keçirərək istifadəçinin etdiyi xətaya uyğun bloku seçir və çap edir.

Istəsəniz bu xətaları bir yerə toplayıb ümumilikdə print(‘xəta oldu’) kimi bir ifadə də əlavə edə bilərsiniz.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    try:
        num=int(input('bir rəqəm yazın:'))
        num1=int(input('bir rəqəm yazın:'))
        print(num/num1)
    except (ValueError,ZeroDivisionError):
        print('Xəta oldu')
```

```
bir rəqəm yazın:12
bir rəqəm yazın:asd
Xəta oldu
bir rəqəm yazın:12
bir rəqəm yazın:0
Xəta oldu
bir rəqəm yazın:
```

try except as   ifadəsi

Bu ifadə,xətanı tamamilə deyil,xətanın hansı tipdə olduğunu bildirir.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

try:
    num=int(input('bir rəqəm yazın:'))
    num1=int(input('bir rəqəm yazın:'))
    print(num/num1)
except ValueError as e:
    print(e)
```

```
bir rəqəm yazın:12
```

```
bir rəqəm yazın:e
invalid literal for int() with base 10: 'e'
>>>
```

Yuxarıda gördüyünüz kimi xəta tamamilə dəyişərək “invalid literal for int() with base 10: 'e' “ kimi çıxdı.

Və ya hər bir xətanı cəmləşdirərək

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    try:
        num=int(input('bir rəqəm yazın:'))
        num1=int(input('bir rəqəm yazın:'))
        print(num/num1)
    except (ValueError,ZeroDivisionError) as e:
        print('Xəta oldu,xəta tipi-{}'.format(e))
```

```
bir rəqəm yazın:12
bir rəqəm yazın:sasd
Xəta oldu,xəta tipi-invalid literal for int() with base 10: 'sasd'
bir rəqəm yazın:12
bir rəqəm yazın:0
Xəta oldu,xəta tipi-division by zero
bir rəqəm yazın:
```

Yuxarıdakı kodlarımızda əlavə etdiyimiz as e ifadəsində e-ni istənilən ifadə ilə əvəz edə bilərsiniz.

Try except else

Bu blok vasitəsilə xətaları ayrı-ayrı çap etmə imkanımız olacaq. Yuxarıda yazdığımız kodu təkrar yazaraq

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    try:
        num=int(input('bir rəqəm yazın:'))
        num1=int(input('bir rəqəm yazın:'))

    except ValueError:
         print('sadəcə rəqəm daxil edin')
    else:
         try:
              print(num/num1)
         except ZeroDivisionError:
              print('0-a bölmək olmaz')
```

əlavələrimizi edək.Biz istifadəçinin başdan ValueError xətasını xeyd etdik.Əgər istifadəçi hərf daxil edərsə(bölmə əməliyyatına keçməmiş)başdan xətanı istifadəçiyə bildirsin və ardından başa dönərək yenidən təkrarlasın sualı.Əgər istifadəçi bölmə əməliyyatında 0-dan istifadə edərsə o zaman else operatoru dövrəyə girəcək və ZeroDivisionError xəta növünü çap edəcəkdir.

Try except finally

Bu blok tipi proqramın xətasını bildirməklə finally hissəsində də ifadələri çap edir.

```
#!/usr/bin/env python
```

```
# -*- coding: utf-8 -*-
try:
     x=int(input('rəqəm yazın:'))
     y=int(input('ikinci rəqəmi yazın:'))
     result=x/y
except ZeroDivisionError:
     print('0-a bölmə xətası')
else:
     print('cavab',result)

finally:
     print('program məcburi çalışdı')
```

```
rəqəm yazın:3
ikinci rəqəmi yazın:0
0-a bölmə xətası
program məcburi çalışdı
>>>
```

raise ifadəsi

Bu ifadə proqram daxilində səhv olmasa belə biz tərəfimizdən veriləcək mesaj tipidir.
Məsələn əgər bölmə əməliyyatı zamanı istifadəçinin ikinci rəqəmi 0-yazanda biz ona bir mesaj verə bilərik.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
     num=int(input('write number:'))
     num1=int(input('write number:'))
     if num1==0:
          raise Exception('please dont write zero!')
     print(num/num1)
```

```
write number:12
write number:6
2.0
write number:56
write number:0
Traceback (most recent call last):
  File "/home/panda/Desktop/fr.py", line 7, in <module>
    raise Exception('please don\'t write zero!')
Exception: please don't write zero!
>>>
```

Yuxarıdakı kodlarımızda istifadəçinin 0-girməsi ilə bir mesaj verdik.

Bu xəta növlərini def funksiyasında daha çox istifadə edəcəyimiz üçün üstündə çox qalmayaraq növbəti bəhsə keçirəm.

String cinsi və metodları

string cinsi

dırnaq işarəsi daxilində istənilən ifadə str cinsinə mənsubdur.

```
>>> a='Python'
>>> type(a)
<class 'str'>
>>>
```

a-ya python ifadəsini atdıq və sonra tipini soruşduq və bizə str(string)tipi olduğunu söylədi.
Amma istəsəniz sayları da dırnaq içində yazıb string cinsinə aid edə bilərsiniz.

```
>>> a='123456'
>>> type(a)
<class 'str'>
>>>
```

Eyni qayda ilə tipini soruşduq və bizə string cins tipi olduğunu söylədi. Bu cinsin başqa bir metodu qapalı mötərizə içində hərfləri sorğuya çəkməkdir.

```
>>> a='Python'
>>> a[0]
'P'
>>> a[1]
'y'
>>> a[3]
'h'
>>> a[-3]
'h'
>>> a='123456'
>>> a[1]
'2'
>>> 'java'[3]
'a'
>>> 'java'[0]
'j'
>>>
>>> i='python'
>>> print(i[:1])
p
>>> print(i[::1])
python
>>> print(i[1:])
ython
>>> print(i[1])
y
>>> print(i[::1])
python
>>> print(i[:3])
pyt
```

```
>>> print(i[1::4])
yn
>>>
```

Son ifadəmizdə 1-ci hərfi və 4-cü hərfdən sonrakını göstər əmri verdik.

```
>>> for i in range(4):
        print('python'[i])


p
y
t
h
>>> for i in range(5):
        print('python'[i])


p
y
t
h
o
>>>
```

yuxarıdakı ifadələrimizdə isə 5-ə qədər yəni p-0 y-1 t-2 h-3 o-4 hərflərini göstər əmri verdik.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
i=input('write you name:')
v=1
for x in range(len(i)):
    print(i[1:4],x+v)
```

```
write you name:rashad
ash 1
ash 2
ash 3
```

```
ash 4
ash 5
ash 6
>>>

>>> url='www.tecnoaz.com'
>>> url[4::12]
't'
>>> url[4:12]
'tecnoaz.'
>>> url[4:11]
'tecnoaz'
>>> url[:3]
'www'
>>> url[:14]
'www.tecnoaz.co'
>>> url[::14]
'wm'
>>> url[::10]
'wz'
>>> url[:13]
'www.tecnoaz.c'
>>> url[::-13]
'mw'
>>> url[::-10]
'mt'
>>>
```

sorted() metodu

metod string cinsinə aid ifadələri(sözləri) əlifba sırasına görə sıralayır.Daha dəqiq,ifadənin daxilindəki hərfləri əlifba sırası ilə düzür.

```
>>> for i in 'python':
	sorted(i)


['p']
['y']
['t']
```

```
['h']
['o']
['n']
>>>
```

Terminaldan

```
>>> dir(str)
```

yazmaqla qarşımıza çıxan metodlar string moduluna mənsubdur.
Və ya istifadə edə biləcəyimiz metodları listələyək.

```
>>> for i in dir(str):
        if '_' not in i:
            print(i)
```

capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill'

Bu bəhsdə istifadə edəcəyimiz metodlar yuxarıdakılardır.Metodları yuxarıdakı ardıcıllıqla öyrənəcəyik.Bu üsulla bir-birinə yaxın funksiyanı yerinə yetirən metodları üstündən zaman keçərək öyrənməyiniz daha məqsədə uyğundur.İlk olaraq
capitalize metodundan başlayaq.

Capitalize metodu

Bu metod ifadənin yalnız ilk hərfini böyüdür.Əgər siz bir neçə ifadə yazsaz,metod yalnızca ilk ifadənin-ilk hərfinə baxacaq.Metod ifadəni soldan-sağa oxuyur.Ifadənin ilk hərfini böyütməklə yanaşı onu ekrana çap edir.(Son cümləni yazmaqda məqsədim odurki irəlidə keçəcəyimiz bəzi

metodlar etdiyi funksiyaları ekrana çap etmir)

```
>>> 'general'.capitalize()
'General'
>>> i='python'
>>> i.capitalize()
'Python'
>>>

>>> i='java','c++','matlab'
>>> i.capitalize()
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    i.capitalize()
AttributeError: 'tuple' object has no attribute 'capitalize'
```

Və gördüyümüz kimi birdən çox ifadə yazdıqda capitalize metodunda xəta aldıq.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
fayl=open('wordlist.txt','a')
print('python'.capitalize(),file=fayl,flush=True)
```

casefold metodu

Bu metod bütünlüklə böyük hərflərdən ibarət bir ifadəni tamamilə kiçik hərflərə çevirir.

```
>>> 'PYTHON'.casefold()
'python'
>>> 'python'.casefold()
'python'
>>> i='AZERBAIJAN'
```

```
>>> i.casefold()
'azerbaijan'
>>>
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
fayl=open('wordlist.txt','a')
print('PYTHON'.casefold(),file=fayl,flush=True)
```

wordlist.txt adlı bir fayl açdıq və casefold metodundan istifadə edərək faylımıza böyük hərflərdən ibarət PYTHON ifadəsini tamamilə kiçildərək yazdırdıq.

center metodu

Dilimizə mərkəz,mərkəzləşdirmək kimi tərcümə olunur.Mənası qədər də python da iş görür.Yəni ifadəni sağ və sol istiqamətlərdən verdiyiniz ölçüdə mərkəzləşdirir.Yalnız bir argument ala bilir.Misallara baxaq.

```
>>> 'roshanna'.center(10)
' roshanna '
>>> 'roshanna'.center(30)
'           roshanna           '
>>> i='python'
>>> i.center(12)
'   python   '
>>> print('algebra'.center(8))
algebra
>>> print('algebra'.center(30))
           algebra
>>> print(i.center(20))
       python
>>>
```

ekran görüntüsündən gördüyümüz kimi ifadə sağ və sol tərəfdən verdiyimiz argument qədər mərkəzləşdi(i.center(20)

count metodu

Dilimizə saymaq kimi tərcümə olunur.Bu metod ifadə daxilində hərf və ya rəqəmin nəqədər olduğunu göstərir.Ekrana çap edir.
Metod 3-sayda argument ala bilir.Misallara baxaq.

```
>>> i='maldivia'
>>> i.count('a')
2
>>>
```

Yuxarıda yazdığımız tək argument olaraq 'a'-hərfinin, 'maldivia' ifadəsi daxilində nəqədər olduğunu soruşduq və bizə 2 -sayını göstərdi.bəli bu doğrudur ifadə daxilində iki a hərfi var.Python sıralama işində 0-dan başlayaraq saymağa başlayır.

0	1	2	3	4	5	6	7
m	a	l	d	i	v	i	a

Tabeldə gördüyümüz kimi maldivia 8-hərfdən ibarət olduğunu görürük,amma python 0-dan başlayaraq sayır.İndidə count metoduna sıra sayı verməklə,həmin sıra sayından başlayaraq sonrakı hərfləri test etsin.

```
>>> i.count('a',1)
2
```

Kodlarımızda 1-rəqəmini yazmağda məqsədimiz a-hərfini maldivia ifadəsində yalnız 1-ci sıradan başlayaraq nəzərdən keçirsin.Və 1 yazmaqla count metodu,maldivia deyil,aldivia ifadəsindən başlayaraq a-hərfinin ifadə daxilində nəqədər olduğunu gözdən keçirəcək.Aşağıdakı kodlarımızda isə ekran görüntüsündə 1 rəqəmini görürük.çünki biz pythona ldivia ifadəsindən başlayaraq a-hərfini gözdən keçirməyi əmr etdik və qarşılığında yalnız son sırada olan a hərfi 1-ədəd olduğundan ekrana bir rəqəmini göstərdi.

```
>>> i.count('a',2)
1
>>>
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
i=input('bir ifadə yazın:')
for x in i:
    print('{} hərfi {} ifadəsində {} dəfə keçir'.format(x,i,i.count(x)))
```

```
bir ifadə yazın:azerbaijan
a hərfi azerbaijan ifadəsində 3 dəfə keçir
z hərfi azerbaijan ifadəsində 1 dəfə keçir
e hərfi azerbaijan ifadəsində 1 dəfə keçir
r hərfi azerbaijan ifadəsində 1 dəfə keçir
b hərfi azerbaijan ifadəsində 1 dəfə keçir
a hərfi azerbaijan ifadəsində 3 dəfə keçir
i hərfi azerbaijan ifadəsində 1 dəfə keçir
j hərfi azerbaijan ifadəsində 1 dəfə keçir
a hərfi azerbaijan ifadəsində 3 dəfə keçir
n hərfi azerbaijan ifadəsində 1 dəfə keçir
>>>
```

encode metodu

Bu metod kodlama metodudur.ifadəni etap-etap kodlarını açaraq son nəticəyə bağlana bilərik.Tək halda işlənə bilməyən metoddur.Aşağıdakı misallara baxaq.

```
>>> original = '27 岁少妇生孩子后变老'
```

Yuxarıdakı ifadəmizin tipini soruşaq.

```
>>> type(original)
<class 'str'>
```

və string cinsinə mənsubdur.

```
>>> encoded = original.encode('utf-8')
>>> print(encoded)
b'27\xe5\xb2\x81\xe5\xb0\x91\xe5\xa6\x87\xe7\x94\x9f\xe5\xad\xa9\xe5\xad\x90\xe5\x90\x8e\xe5\x8f\x98\xe8\x80\x81'
```

ifadəmizin utf-8 görünüşü yuxarıdakı kimidir

```
>>> type(encoded)
<class 'bytes'>
>>> encoded2 = bytes(original, 'utf-8')
>>> print(encoded2)
b'27\xe5\xb2\x81\xe5\xb0\x91\xe5\xa6\x87\xe7\x94\x9f\xe5\xad\xa9\xe5\xad\x90\xe5\x90\x8e\xe5\x8f\x98\xe8\x80\x81'
>>> type(encoded2)
<class 'bytes'>
>>> print(encoded+encoded2)
b'27\xe5\xb2\x81\xe5\xb0\x91\xe5\xa6\x87\xe7\x94\x9f\xe5\xad\xa9\xe5\xad\x90\xe5\x90\x8e\xe5\x8f\x98\xe8\x80\x8127\xe5\xb2\x81\xe5\xb0\x91\xe5\xa6\x87\xe7\x94\x9f\xe5\xad\xa9\xe5\xad\x90\xe5\x90\x8e\xe5\x8f\x98\xe8\x80\x81'
>>> decoded = encoded.decode('utf-8')
>>> print(decoded)
27 岁少妇生孩子后变老
>>> decoded2 = str(encoded2, 'utf-8')
>>> print(decoded2)
27 岁少妇生孩子后变老
>>>
```

və kodlarımızın açıqlamasını əldə etdik.

endswith() metodu

Bu metod yardımı ilə ifadə sonunun göstərə bildiyimiz argumentlə bitdiyini soruşa bilərik.
Misallara baxaq

```
>>> 'python'.endswith('a')
False
>>> 'python'.endswith('n')
True
>>>
```

Ilk ifadədə False yəni inkar aldıq.Çünki python sözünün sonu a ilə bitmir.İkinci ifadəmizdə isə True yəni təsdiq aldıq,bəli python sözünün sonu n-ilə bitir.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
list=['adale.mp3','python.py','doc.txt','geany.py']
for i in list:
    if i.endswith('py'):
        print(i)
```

```
python.py
geany.py
>>>
```

Və sonu py ilə bitən ifadələri sorğuya çəkdik.
Və ya

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
a='python.py','adele.mp3','brew.txt','wordlist.txt'
for i in a:
    if i.endswith('txt'):
        print(i)
```

```
brew.txt
wordlist.txt
>>>
```

expandtabs() metodu

dilimizə genişlətmək,genişləndirmək kimi tərcümə olunur.Metodun pythonda roluna baxaq

```
>>> 'azerbaijan\trepublic'.expandtabs(15)
'azerbaijan     republic'
>>>
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
a='python.py\tadele.mp3\tbrew.txt\twordlist.txt'
print([a.expandtabs(15)])
```

```
['python.py      adele.mp3      brew.txt       wordlist.txt']
>>>
```

metod ifadələr arasında məsafələri verdiyimiz argument qədər artırır.

find()   metodu

Bu metod ifadə daxilində soruşduğumuz argumentin neçənci sırada dayandığını çap edir.Əgər axtardığımız argument ifadə daxilində yoxdursa ekrana -1 çap edir.
Metoda dair misallara baxaq

```
>>> 'python'.find('h')
3
```

argument olaraq 'h' soruşduq və bizə 3-cü sırada dayandığını söylədi.Öncəki bəhslərdə qeyd etdiyim kimi python ilk 0-dan başlayaraq sayır.(Bəzi metodlarda istisnalar olacaq)

```
>>> '123456'.find('3')
2
>>>
```

Eyni qayda ilə sorğuya çəkdik və 3-rəqəminin 2-ci sırada dayandığını söylədi.Qeyd edimki sətir və sıranı(sütun) qarışdırmayın.sətir soldan aşağıya doğru,sıra isə yuxarıdan sağa doğru sayılan parametrlərdir.

format()   metodu

Bu metoda öncəki bəhslərdə baxmışdıq.Təkrar etməyin ziyanı olmadığı üçün misallara baxaq.

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
```

```
'a, b, c'

>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'

>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'

>>> '{2}, {1}, {0}'.format(*'abc')
'c, b, a'

>>> '{0}{1}{0}'.format('abra', 'cad')
'abracadabra'

>>> 'Koordinatlar: {latitude}, {longitude}'.format(latitude='37.24N',
longitude='-115.81W')
'Koordinatlar: 37.24N, -115.81W'

>>> kord = {'latitude': '37.24N', 'longitude': '-115.81W'}
>>> 'Koordinatlar: {latitude}, {longitude}'.format(**kord)
'Koordinatlar: 37.24N, -115.81W'

>>> koord = (3, 5)
>>> 'X: {0[0]};  Y: {0[1]}'.format(koord)
'X: 3;  Y: 5'
```

index() metodu

Bu metodun yardımı ilə list,tuple və dictionary daxilində istədiyimiz ifadənin axtarışına çıxa bilərik.

```
>>> i='electronic','techaz','dnsserver'
>>> i.index('techaz')
1
>>>

>>> list=['electronic','techaz','dnsserver']
>>> list.index('dnsserver')
2
```

ifadəni istifadəçidən alaraq axtarış edək

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    try:
        i=input('write something:')
        list=['electronic','techaz','dnsserver','book','underground','alias']
        if list.index(i):
            print (True,i)
    except ValueError:
        print(False,i)
```

```
write something:df
False df
write something:rt
False rt
write something:alias
True alias
write something:
```

isalnum()   metodu

Bu metod ifadənin hərf və rəqəmlərdən ibarət olduğunu təyin edir. Hərf və rəqəmlərdən təşkil olunan parametr alphanum adlanır.(yəni abcdf123)
Misallara baxaq

```
>>> 'asd123'.isalnum()
True
>>> i='cvfd45fa'
>>> i.isalnum()
True
>>> 'python*9'.isalnum()
```

```
False
>>>
```

isalpha() metodu

Bu metodun yardımı ilə ifadənin tamamilə hərflərdən ibarət olduğu müəyyən edilir.

```
>>> 'python'.isalpha()
True
>>> i='python'
>>> i.isalpha()
True
>>> 'python34'.isalpha()
False
>>>
```

isdecimal() metodu

Bu metod ifadənin onluq say sistemi olduğunu müəyyən edir.

```
>>> 'python12'.isdecimal()
False
>>> '120'.isdecimal()
True
>>> '1298012'.isdecimal()
True
>>>
```

isdigit()   metodu

Metod ifadənin tamamilə saylardan ibarət olmasını müəyyən edir.

```
>>> '123'.isdigit()
```

```
True
>>> '12343d'.isdigit()
False
>>>
```

isidentifier()   metodu

i='electronic' gördüyünüz bu ifadə ayrı-ayrılıqda i və electronic söz və hərfindən ibarətdir.Bəzi metodlarda istifadəsi rahat olsun deyə i-hərfinə dırnaq içində söz yerləşdirmişik.Pythonda ürəyimiz istəyən hərflərə,hərf birləşmələrinə və ya sözlərə ifadə yerləşdirə bilmərik.Pythonun sabit ifadələri aşağıdakılardır.

```
>>> from keyword import*
>>> print(kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
>>>
```

Yuxarıda gördüyünüz ifadələri funksiya kimi istifadə edə bilərik.Amma

```
>>> as='python'
SyntaxError: invalid syntax
>>> for='elsever'
SyntaxError: invalid syntax
>>>
```

kimi yazsaq xəta alacıq.Bizim metodumuzda bu cür xətaların öncədən baş verməməsi üçün istifadə edəcəyimiz ifadəni metodumuz vasitəsilə sorğuya çəkib,istifadə edə biləcəyimizi öyrənə bilərik.

```
>>> 'a1'.isidentifier()
True
>>> '1a'.isidentifier()
False
>>> '-b'.isidentifier()
False
```

```
>>> 'b-'.isidentifier()
False
>>>
```

Qeyd edimki pythonun sabit ifadələrini sorğuya çəkməyin,çünki bu metod onlara true verəcək.Ona görəki biz ifadəmizi dırnaq içində yazırıq.Sabit ifadələrdən başqa istənilən ifadəni sorğuya çəkə bilərsiniz.

islower()   metodu

Bu metod vasitəsilə ifadənin kiçik hərflərdən ibarət olması müəyyən edilir.

```
>>> 'PYTHON'.islower()
False
>>> 'python'.islower()
True
>>> 'Python'.islower()
False
>>> i='algida'
>>> i.islower()
True
>>>
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    passw=input('parolunuzu yazın:')
    if not passw.islower():
        print('zəhmət olmasa kiçik hərflərdən istifadə edin!')
    else:
        print('parolunuz qəbul olundu!')
```

```
parolunuzu yazın:alfave
parolunuz qəbul olundu!
parolunuzu yazın:Python
zəhmət olmasa kiçik hərflərdən istifadə edin!
parolunuzu yazın:
```

isnumeric() metodu

Bu metod vasitəsilə ifadənin tamamilə saylardan ibarət olub-olmadığı təyin olunur.Cavab olaraq True(təsdiq),False(inkar) verir.

```
>>> '1234'.isnumeric()
True
>>> 'phg65'.isnumeric()
False
>>> i='1234'
>>> i.isnumeric()
True
>>>
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    passw=input('parolunuzu yazın:')
    if not passw.isnumeric():
        print('yalnız rəqəmlərdən istifadə edin ')
    else:
        print('parolunuz qəbul olundu!')
```

```
parolunuzu yazın:343534
parolunuz qəbul olundu!
parolunuzu yazın:gdg
yalnız rəqəmlərdən istifadə edin
parolunuzu yazın:242
parolunuz qəbul olundu!
parolunuzu yazın:ege4353
yalnız rəqəmlərdən istifadə edin
```

isprintable() metodu

əvvəlki bəhslərdə keçdiyimiz isidentifier() metoduna bənzər metoddur.Bu metod ifadənin istifadə ola biləcəyini true ifadəsi ilə çap edir.Əgər xüsusi işarəni sorğuya çəksək metod false olaraq verəcək.

```
>>> 'python'.isprintable()
True
>>> '\a'.isprintable()
False
>>> '\c'.isprintable()
True
>>>
```

Gördüyümüz kimi xüsusi işarələrdən istifadə etdikdə metod False(inkar) verir.Digər ifadələrdə isə True(aldıq) verir.

isspace() metodu

adından da göründüyü kimi space kəliməsi keçir metod daxilində.Metod ifadənin bütünlüklə boşluqlardan ibarət olmasını müəyyən edir.

```
>>> '  '.isspace()
True
>>> '  45'.isspace()
False
>>> '23  '.isspace()
False
>>>
```

istitle() metodu

Bu metod yardımı ilə ifadə daxilindəki sözlərin ilk hərfi böyük olub-olmadığı müəyyən edilir.

```
>>> i='Python Is Very'
>>> i.istitle()
True
>>>
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
str='Biz hər gün Göygölün sahilinə gedərdik'
print(str.istitle())
```

```
False
>>>
```

metod bütünlüklə ifadədə olan sözlərin baş hərfini nəzərdən keçirir,əgər sözün biri kiçik hərflərdən təşkil olunubsa False (inkar) verəcək.

issupper() metodu

upper sözü böyük,yuxarı mənasını verir.Metod ifadənin bütünlüklə böyük hərflərdən təşkil olub-olmadığını müəyyən edir.Cavabı iki parametr- True və False

```
>>> 'PYTHON'.isupper()
```

```
True
>>>
>>> i='algoritma'
>>> i.isupper()
False
>>> i='ALGORITMA'
>>> i.isupper()
True
>>>
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
str='BIZ HƏR GÜN GÖYGÖLÜN KƏNARINA GEDƏRDİK'
print(str.isupper())
```

```
True
>>>
```

join() metodu

Bu metod ayrı-ayrı ifadələri toplayaraq bir cümlə formasına salır.Metod bir argument ala bilir.

```
>>> i='python','hello','py'
>>> ''.join(i)
'pythonhellopy'
>>>
```

Gördüyümüz kimi ayrı-ayrı ifadələrimizi bir yerə topladı.Bizə maraqlı tərəfi başda boşluq qoymayaraq metodumuzu yazmağımız.
Və ya

```
>>> list=['else','elif','if']
>>> i=''
>>> i.join(list)
'elseelifif'
>>>
```

Metod biraz sep metoduna bənzəyir.Biz kodlarımızda boşluq qoymayaraq ifadələrimizi birləşdirdik və bərbad göründü.İndi digər işarələrlə əvəz edərək yazaq

```
>>> list
['else', 'elif', 'if']
>>> i=' '
>>> i.join(list)
'else elif if'
>>> i='-'
>>> i.join(list)
'else-elif-if'
>>> i='*_*'
>>> i.join(list)
'else*_*elif*_*if'
>>>
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
url='www','niim','az'
print('.'.join(url))

www.niim.az
>>>
```

Hətda bu metodda iki və daha artıq string argumenti vermək olur.Son yazdığımız adresdə hər ifadə arasına nöqtə qoyduqsa indi də müsbət(+) işarəsindən istifadə edərək ünvanın əvvəlinə və ifadələr arasına argumentlər əlaə edək.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
url='www','niim','az'
print('http://'+'.'.join(url))

http://www.niim.az
>>>
```

Gördüyünüz kimi ilk yazdığımız argument 'http://'   -ifadənin başına,'.'

-argumenti isə ifadələrin arasına daxil oldu.

Digər işarələrdən də istifadə edərək metodumuza tətbiq etdik.

ljust() metodu

Metod iki argument alır.ifadəni sola sıxışdıraraq verdiyimiz argumenti ifadənin sonundan yerləşdirir.Baş hərfi l-olması ingilis dilinin left kəliməsinə bağlıdır.(just-sadəcə,yalnız)Yəni sadəcə sola kimi ifadə edək.Verdiyiniz argument əgər ifadənizdən az saydadırsa o zaman çapda görünməyəcək.Yəni

```
>>> i.ljust(4,'.')
'Stanislav'
```

ifadəmiz 9 hərfdən ibarət olduğu üçün yazdığımız 4 sayında nöqtələr görünmür.

1	2	3	4	5	6	7	8	9
S	t	a	n	i	s	l	a	v

Və cədvəldən göründüyü kimi 9 hərfli ifadəyə 4 argumenti uyğun gəlmir.Biz 10-argumentini yoxlayaraq buna əmin olaq

```
>>> 'Stanislav'.ljust(10,'-')
'Stanislav-'
>>>
```

Bəli,gördüyünüz kimi 9-hərfli ifadəmizdən sonra yalnız bir xətt çap oldu.Və belə nəticəyə gəldikki ljust metodu aldığı argumenti ifadənin ilk hərfindən başlayaraq sayır,əgər argument sayı ifadə sayından artıqdırsa,ifadənin sağından verdiyimiz işarə tipini yerləşdirəcək.Yox əgər

azdırsa heç bir dəyişiklik etmədən sadəcə ifadəni ekrana çap edəcək.

```
>>> i.ljust(15,'-')
'Stanislav------'
```

və son yazdığımız misalda stanislav 9 hərf verdiyimiz argument sayı 15-dir.15-9=6 və 6 sayda tire işarəsini Stanislav sözünün ardına əlavə etdi.

lower() metodu

Dilimizə aşağı kimi tərcümə olunsada ifadələrlə bağlı metod,böyük hərflə yazılan ifadələri,kiçik hərflərə çevirir
Misallara baxaq.

```
>>> 'Python'.lower()
'python'
>>> 'PYTHON'.lower()
'python'
>>> 'python'.lower()
'python'
>>> '1234'.lower()
'1234'
>>>
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    parol=input('write you passw:')
    if  parol== parol.lower():
        print('successfully')
    else:
        print('try again')
```

```
write you passw:python
Successfully
write you passw:PYthon
try again
write you passw:
```

maketrans(), translate()   metodları.

Bu iki metod daima bir yerdə işlədilir.translate() metodu dilimizə tərcümə etmək,çevirmək kimi tərcümə olunur.
Əlimizdə əlifba var
alfabe='abcdefyhiklmnopqrstvx' latın əlifbası
bu əlifbaya uyğun azərbaycan hərflərini qarşılıqlı yazaq

```
alfabe='abcdefyhiklmnopqrstvx'
alf_az='abçdəfğyhiklmnöpqrştvx'
```

Yuxarıda əlifbada qarşılıq gələn hərfləri yazdıq.İndidə istifadəçidən bir mətn alaraq kodlarımızı tətbiq edək.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
alfabe='abcdefgyhiklmnopqrstvx'
alf_az='abçdəfğyhiklmnöpqrştvx'
text=input('mətn yazın:')
convert=text.maketrans(alfabe,alf_az)
print(text.translate(convert))
```

```
mətn yazın:biz hemise olkemizde azad ve demokratik yasamagi ustun tuturuq
biz həmişə ölkəmizdə azad və dəmökratik yaşamaği uştun tuturuq
>>>
```

Və kodlarımızda istifadəçidən mətni alırıq.Daha sonra bu metni alfabe və alfa_az hərf qarşılığı ilə çeviririk,sonra print daxilinə translate metodu ilə bu çevirməni çap edərək

```
mətn yazın:biz hemise olkemizde azad ve demokratik yasamagi ustun tuturuq
biz həmişə ölkəmizdə azad və dəmökratik yaşamaği uştun tuturuq
>>>
```

belə bir cümlə əldə edirik.Gördüyümüz kimi bəzi hərflər xətalıdır.Əslində burda heç bir xəta yoxdur,çünki hərflərin qarşılaşdırma metodunu biz özümüz yazmışıq e-yə ə-o-ya ö -verdiyimiz üçün dəmökratik ifadəsini aldıq.Əgər sonda print(convert) yazsaq

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
alfabe='abcdefgyhiklmnopqrstvx'
alf_az='abçdəfğyhiklmnöpqrştvx'
text=input('mətn yazın:')
convert=text.maketrans(alfabe,alf_az)
print([convert])
```

```
mətn yazın:azerbaijan deniz akademiyasi
[{97: 97, 98: 98, 99: 231, 100: 100, 101: 601, 102: 102, 103: 287, 104: 104, 105: 105, 107: 107, 108: 108, 109: 109, 110: 110, 111: 246, 112: 112, 113: 113, 114: 114, 115: 351, 116: 116, 118: 118, 120: 120, 121: 121}]
>>>
```

qarşınıza saylar çıxacaq.Yuxarıdakı rəqəmlərə açıqlama verək.Yuxarıdakı rəqəmlər hər biri bir hərfi ifadə edir.Bu rəqəmlərin qarşılıqlı hansı hərfə uyğun gəldiyini təyin etmək üçün chr() adlı funksiyadan istifadə edəcik.chr() funksiyası haqqında irəlidəki bəhslərimizdə geniş danışacağıq.

```
>>> chr(107)
'k'
>>> chr(120)
'x'
>>>
```

Və ekran çapından məlum olduğu kimi 107-ə k hərfi,120-ə isə x hərfi uyğun gəlir.

For operatorundan istifadə edərək qarşılıqlı hərf-rəqəmlərə baxaq.

```
>>> for i in 97,110,118,120,121,114,246:
        print(chr(i))


a
n
v
x
y
r
ö
>>>
```

və ya

```
>>> for i in 97,110,118,120,121,114,246:
        print(i,chr(i))


 97  a
110 n
118 v
120 x
121 y
114 r
246 ö
>>>
```

saylara qarşılıqlı hərfləri görə bilərik.

partition()   metodu

Dilimizə hissə,parçalara kimi tərcümə olunur.Metod, ifadəni istənilən yerindən parçalara ayıra bilir.Misallara baxaq

```
>>> 'Vladivastok'.partition('iv')
('Vlad', 'iv', 'astok')
>>> i='Azerbaijan'
>>> i.partition('ba')
('Azer', 'ba', 'ijan')
>>>
```

replace()   metodu

Dilimizə ‘əvəz etmək’ kimi tərcümə olunur.pythonda da metod tərcüməsi qədər iş görür.
Yəni bir ifadə daxilində argumenti digəri ilə əvəz edir.

```
>>> 'python'.replace('p','P')
'Python'
>>> 'azerbaijan'.replace('e','ə')
'azərbaijan'
>>>
>>> ' ango'.replace(' ','T')
'Tango'
>>>
```

rfind()   metodu

Metod ifadə daxilindəki argumentin sırasını müəyyən edir.Amma bu metod sağdan oxuyaraq qiymətləndirir.Baş hərfinə r-yəni right sözündən götürülüb.Metod əgər argumenti tapmayarsa o zaman ekrana mənfi bir (-1) verəcək.

```
>>> i='apacalipso'
>>> i.rfind('o')
9
>>> i.rfind('i')
```

```
6
>>> i.rfind('a')
4
>>>
>>> 'python'.rfind('a')
-1
>>>
```

Son ifadədə olduğu kimi mənfi bir ifadəsi həmin argumenti tapmadığına dair xətadır.Əlbətdə python sözündə a-hərfi yoxdur.

rindex()   metodu

Metod ifadə daxilində argumentləri müəyyən edir.Baş hərfi r-olduğuna görə ifadəni sağdan-sola oxuyur.Metod əgər argumanti müəyyən etməyərsə o zaman ValuError xətası verəcək.

```
>>> 'python'.rindex('a')
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    'python'.rindex('a')
ValueError: substring not found
>>>

>>> 'python'.rindex('p')
0
>>>
```

rjust()   metodu

Metod ifadəni sağa sıxışdıraraq sol tərəfdən verdiyimiz argumentləri daxil edir.Metod ljust metoduna bənzəyir.Baş hərfi r- yəni right sağa yönəlir.Iki argument alır.

```
>>> 'python'.rjust(8,'-')
'--python'
>>>
```

rpartition()   metodu

Metod ifadəni bir neçə hissələrə ayırır.İfadəni sağdan-sola oxuyaraq verdiyimiz argumentə görə bölgü aparır.

```
>>> 'azerbaijan'.rpartition('a')
('azerbaij', 'a', 'n')
```

Gördüyünüz kimi ilk a hərfdən bölgünü deyil sağdan oxuyaraq hissələrə böldü.

```
>>> 'azerbaijan deniz akademiyasi'.rsplit(' ',1)
['azerbaijan deniz', 'akademiyasi']
>>> 'azerbaijan deniz akademiyasi'.rsplit('.',2)
['azerbaijan deniz akademiyasi']
>>> 'azerbaijan deniz akademiyasi'.rsplit(' ',0)
['azerbaijan deniz akademiyasi']
>>> 'azerbaijan deniz akademiyasi'.rsplit(' ',2)
['azerbaijan', 'deniz', 'akademiyasi']
>>>
```

rstrip()   metodu

Metod ifadə daxilində son son hərfi(və ya rəqəmi) silir.baş hərfi r- olmasından bizə məlumdurki metod ifadəni sağdan-sola oxuyaraq

funksiyasını yerinə yetirir.Misallara baxaq.Bir argument alır.

```
>>> 'python'.rstrip('n')
'pytho'
>>> i='django'
>>> i.rstrip('o')
'djang'
>>> 'python'.rstrip('o')
'python'
```

Son ifadəmizdə biz sondan ikinci hərfi silməsini tələb etsəkdə bu baş tutmadı.

split()   metodu

Metod ifadə daxilində sözləri hissələrə ayırır.

```
>>> 'Bakı dövlət universiteti'.split()
['Bakı', 'dövlət', 'universiteti']
>>> for i in 'Bakı dövlət universiteti'.split():
        print(i)



Bakı
dövlət
universiteti
>>>
```

Əgər sözlər arasına hər hansı bir işarə yazsanış metod sözləri ümumilikdə dırnaq içinə alacaq

```
>>> 'Django,Python,Java'.split()
['Django,Python,Java']
>>>
```

əgər for operatorundan istifadə edərək ifadələri bir sətrə yerləşdirmək istəsəniz

```
>>> for i in 'Django,Python,Java'.split('1'):
	print(i)


Django,Python,Java
>>>
```

splitlines() və True

Metodumuz uzun bir mətn sistemlərində daha çox istifadə olunur,və True parametrini verərək mətnlərdə olan cümlələri sətrlərə bölür,eləcədə böldüyü sətrlərin sonuna xüsusi keçid işarələrini göstərir.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
text="""Python is a widely used high-level, general-purpose,
interpreted, dynamic programming language.[24][25] Its design
philosophy emphasizes code readability, and its syntax allows
programmers to express concepts in fewer lines of code than
possible in languages such as C++ or Jav"""
print(text.splitlines())

['Python is a widely used high-level, general-purpose,', 'interpreted, dynamic programming language.[24][25] Its design', 'philosophy emphasizes code readability, and its syntax allows', 'programmers to express concepts in fewer lines of code than', 'possible in languages such as C++ or Jav']
>>>
```

Indi gəlin True parametri ilə yoxlayaq

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
text="""Python is a widely used high-level, general-purpose,
interpreted, dynamic programming language.[24][25] Its design
philosophy emphasizes code readability, and its syntax allows
programmers to express concepts in fewer lines of code than
possible in languages such as C++ or Jav"""
print(text.splitlines(True))
```

```
['Python is a widely used high-level, general-purpose,\n', 'interpreted, dynamic programming language.[24][25] Its design\n', 'philosophy emphasizes code readability, and its syntax allows\n', 'programmers to express concepts in fewer lines of code than\n', 'possible in languages such as C++ or Jav']
>>>
```

Və True parametri ilə harda xüsusi işarələr işlədə biləcəyimizə dair metod bizə yardım etdi.

startswith()   metodu

metod ifadənin baş tərəfinə baxır,mənasından da başa düşdüyümüz qədər start-yəni başlamaq,başlanğıc kimi ifadənin ilk hissəsini oxuyur.misallara baxaq.

```
>>> 'range'.startswith('p')
False
```

```
>>> 'range'.startswith('r')
True
>>> i='avakado'
>>> i.startswith('a')
True
>>>
```

ilk başda range sözündə ilk hərf r-ilə başladığı üçün biz False parametrini aldıq,çünki r-deyil biz p-hərfinin olub-olmadığını sorğuya çəkdik.

strip() metodu

Metod ifadənin kənarlarında olan xüsusi işarələri və boşluqları kəsib atar,eləcədə biz argument verərək ifadənin özünə də müdaxilə edə bilərik.Misallara baxaq

```
>>> ' elevator'.strip()
'elevator'
```

heç bir argument vermədən metod ilk başdakı boşluğu kəsib atdı

```
>>> '**elevator'.strip()
'**elevator'
```

burda isə ulduz işarələrin də ifadə ilə bərabər hesab edərək heç bir dəyişiklik etmədi,belə hallarda isə biz metoda argument verərək silə bilərik.

```
>>> '**elevator'.strip('**')
'elevator'
```

```
>>> 'elevator   '.strip()
'elevator'
```

```
>>>

>>> '\nelevator'.strip()
'elevator'
>>>
```

və gördüyünüz kimi xüsusi işarəmiz olan \n -i kəsib atdı.metodun kəsib atdığı xüsusi işarələri bir cədvələ yerləşdirək.

‘ ‘	\t	\n	\r	\v	\f

```
>>> '\uelevator'.strip()
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 0-2: truncated \uXXXX escape
>>> '\Nelevator'.strip()
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 0-1: malformed \N character escape
>>>
```

Gördüyümüz kimi digər xüsusi işarələrdən istifadə etdikdə xəta alırıq.Yuxarıda qeyd etdiyim xüsusi işarələr strip() metodu üçün keçərlidir.

swapcase()   metodu

Metod,əgər ifadə kiçik hərflərdən təşkil olunubsa-ifadəni böyüdür,yox əgər böyük hərflərdən təşkil olunubsa-ifadəni kiçildir.

```
>>> 'python'.swapcase()
'PYTHON'
>>> 'rəşad'.swapcase()
'RƏŞAD'
>>> 'PyThon'.swapcase()
'pYtHON'
>>> 'inzibati xətalar'.swapcase()
```

```
'INZIBATI XƏTALAR'
>>>
```

Metodla bağlı Azərbaycan-əlifba problemi.Axırıncı ifadədən gördüyümüz kimi böyük İ-yerinə böyük I-çap etdi.Bu problemi aradan qaldırmaq üçün replace() metodundan istifadə edəcik.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
text='inzibati xətalar'
for i in text:
    if i=='İ':
        text=text.replace('İ','i')
    elif i=='i':
        text=text.replace('i','İ')
    else:
        text=text.replace(i,i.swapcase())
print(text)

İNZİBATİ XƏTALAR
>>>
```

və ya ifadəmizi böyük hərflərlə yazaraq

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
text='İNZİBATİ XƏTALAR'
for i in text:
    if i=='İ':
        text=text.replace('İ','i')
    elif i=='i':
        text=text.replace('i','İ')
    else:
        text=text.replace(i,i.swapcase())
print(text)

inzibati xətalar
>>>
```

title() metodu

Metod İfadə daxilində bir neçə sözün ilk hərfini böyüdür.

```
>>> 'random randit'.title()
'Random Randit'
>>> 'Random event'.title()
'Random Event'
>>> i='epsi78lon aptitude'
>>> i.title()
'Epsi78Lon Aptitude'
>>> lst=['python','java','eagle']
>>> lst.title()
```

upper() metodu

Metod,ifadəni bütünlüklə böyük hərflərə çevirir.

```
>>> 'python'.upper()
'PYTHON'
>>> 'pYTHON'.upper()
'PYTHON'
>>>
```

zfill() metodu

Metod,z-zero fill-doldurmaq mənasını ifadə edir.Yəni 0-ilə doldur.metod 0-sayını yalnız sol tərəfə doldurur.

```
>>> '34'.zfill(4)
'0034'
>>>
```

yuxarıdakı metod daxilindəki argumentə biz 2-sayını yaza bilmərik,çünki

ifadəmiz iki rəqəmlidir.Metod da sağdan başlayaraq sayır və yerdə qalan boş yerə 0-ı artırır.Bunu yalnız range,random metodların istifadə etsəniz edə biləcəksiniz.

```
>>> for i in range(14):
	print(str(i).zfill(2))


00
01
02
03
04
05
06
07
08
09
10
11
12
13
>>>
```

List

Boş bir list yaradaq.

```
>>> list=[]
>>> type(list)
<class 'list'>
>>>
```

Listəyə ifadələr yerləşdirək.

```
>>> list=['Bakı', 'Gəncə' ,'Sumqayıt', 'Şəki']
>>> list
['Bakı', 'Gəncə', 'Sumqayıt', 'Şəki']
>>> for i in list:
```

```
        print(i)


Bakı
Gəncə
Sumqayıt
Şəki
>>> len(list)
4
>>> max(list,key=len)
'Sumqayıt'
>>> min(list,key=len)
'Bakı'
>>>
```

string cinslərində olduğu kimi listlərdə də ifadələri

```
>>> list[0]
'Bakı'
>>> list[1]
'Gəncə'
>>> list[2]
'Sumqayıt'
>>>
```

çap etmək olur.

```
>>> print(*list[:1])
Bakı
>>> print(*list[::1])
Bakı Gəncə Sumqayıt Şəki
>>> print(*list[::1],sep=',')
Bakı,Gəncə,Sumqayıt,Şəki
>>>
```

List ifadələrini dəyişdirmək

```
>>> list
```

```
['Bakı', 'Gəncə', 'Sumqayıt', 'Şəki']
>>> list[1]='England'
>>> list
['Bakı', 'England', 'Sumqayıt', 'Şəki']
>>>
```

yuxarıdakı kodlarımızda 1-ci sırada dayanan Gəncə ifadəmizin üzərinə England ifadəsini əlavə etdik.Yəni Gəncə ifadəsini dəyişdirdik.
Listlərin say ardıcıllığına nəzər yetirək

0	1	2	3
Bakı	Englan d	Sumqayıt	Şəki

Listlərdə də python saymağa 0-dan başlayır.
Əgər 3-sayından yuxarı ədəd yazaraq ifadə daxil etmək istəsək xəta alacıq.

```
>>> list[4]='python'
Traceback (most recent call last):
  File "<pyshell#96>", line 1, in <module>
    list[4]='python'
IndexError: list assignment index out of range
>>>
```

Kodlarımıza birdən çox ifadə əlavə etmək istəsək

```
>>> list[0:4]='Python','Java','C++','Perl'
>>> list
['Python', 'Java', 'C++', 'Perl']
>>>
```

list[0:4] -kodumuzun nəzəri açıqlaması,0 və 4-ə qədər (0,1,2,3) sıralara,'Python','Java','C++','Perl'   ifadələrini əlavə et.

Əgər

```
>>> list[0:5]='Python','Java','C++','Perl'
>>> list
['Python', 'Java', 'C++', 'Perl']
```

kimi yazsaq belə xəta almayacıq.Çünki listdə dəyişdirdiyimiz yeni ifadələr 0-sırasından başlayaraq listi doldurur.Əgər sıra sayı dəyişdirəcəyimiz ifadələrdən məhdud sayda olarsa o zaman xəta alacağıq.

```
>>> list[0::3]='Python','Java','C++','Perl','C','language'
Traceback (most recent call last):
  File "<pyshell#114>", line 1, in <module>
    list[0::3]='Python','Java','C++','Perl','C','language'
ValueError: attempt to assign sequence of size 6 to extended slice of size 2
>>>
```

Xətada deyildiyi kimi biz sıra sayını 0,1,2 daxil olmaqla məhdudlaşdırıb 6-sayda dəyişən ifadə daxil etdik.

```
>>> list[0::]='Python','Java','C++','Perl'
>>> list
['Python', 'Java', 'C++', 'Perl']
```

Son kodlarımızda isə 0-dan sağa doğru dəyişən ifadələri əlavə et əmri verdik.

```
>>> list[0]='Python','Java','C++','Perl'
>>> list
[('Python', 'Java', 'C++', 'Perl'), 'Java', 'C++', 'Perl']
>>> list[0::]='Python','Java','C++','Perl','C','language'
>>> list
['Python', 'Java', 'C++', 'Perl', 'C', 'language']
```

Əgər tək sıra sayı bildirsək

```
>>> list='exec'
>>> list
'exec'
>>> list[0]='Python','Java','C++','Perl'
Traceback (most recent call last):
  File "<pyshell#118>", line 1, in <module>
    list[0]='Python','Java','C++','Perl'
TypeError: 'str' object does not support item assignment
```

```
>>>
```

Həcm etibarı ilə bərabər olmadığı üçün xəta aldıq.

Bu bəhsdə list daxilində təyin etdiyimiz ifadələri dəyişərək başqa ifadələrlə əvəz etməyi öyrəndik.Növbəti bəhsimizdə isə list daxilindəki ifadələrə toxunmadan daha başqa ifadələri əlavə etməyi öyrənəcəyik.

Listlər-ə ifadələr əlavə etmək

İlk əvvəl bir list yaradaq

```
>>> list=['django','python','tkinter','PyQt',12]
>>> list
['django', 'python', 'tkinter', 'PyQt', 12]
>>>
```

Listlərə ifadələr əlavə etmək üçün müsbət(+)-işarəsindən istifadə edəcəyik.

```
>>> list+[14]
['django', 'python', 'tkinter', 'PyQt', 12, 14]
>>> list+['quadrotor']
['django', 'python', 'tkinter', 'PyQt', 12, 'quadrotor']
>>>
```

müsbət işarəsindən istifadə etməklə əlavə edəcəyimiz ifadəni qapalı mötərizə içərisində göstərərək yerinə yetirdik və birbaşa list,daxilindəki ifadələri eləcədə yeni əlavə olunanları ekrana çap etdi

İfadələri əlavə edərkən qarşılaşa biləcəyiniz xətalara da nəzər yetirək

```
>>> list+'install'
Traceback (most recent call last):
  File "<pyshell#123>", line 1, in <module>
    list+'install'
TypeError: can only concatenate list (not "str") to list
>>>
```

ifadəmizi qapalı mötərizə içində göstərmədiyimiz üçün xəta aldıq.Bu tək string cinslərinə aid deyil

```
>>> list+45
Traceback (most recent call last):
  File "<pyshell#124>", line 1, in <module>
    list+45
TypeError: can only concatenate list (not "int") to list
>>>
```

Eyni xətanı integer cinsində də aldıq.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    list=[]
    append=input('ifadəni daxil edin>>> ')
    i=list+[append]
    print(i)
    print('daxil etdiyiniz ifadə: %s,tipi:%s'%(append,type(append)))
```

```
ifadəni daxil edin>>> python
['python']
daxil etdiyiniz ifadə: python,tipi:<class 'str'>
ifadəni daxil edin>>> Java
['Java']
daxil etdiyiniz ifadə: Java,tipi:<class 'str'>
ifadəni daxil edin>>> programming
['programming']
daxil etdiyiniz ifadə: programming,tipi:<class 'str'>
ifadəni daxil edin>>> 8
['8']
daxil etdiyiniz ifadə: 8,tipi:<class 'str'>
ifadəni daxil edin>>>
```

%s -parametrini format() metodu ilə əvəz edərək

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
```

```
    list=[]
    append=input('ifadəni daxil edin>>> ')
    i=list+[append]
    print(i)
    print('daxil etdiyiniz ifadə: {},tipi:{}'.format(append,type(append)))
```

```
ifadəni daxil edin>>> epsilon
['epsilon']
daxil etdiyiniz ifadə: epsilon,tipi:<class 'str'>
ifadəni daxil edin>>> balloon
['balloon']
daxil etdiyiniz ifadə: balloon,tipi:<class 'str'>
ifadəni daxil edin>>>
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    list=[]

    i=input('write something:')
    if len(i)<=4:
        print('%s ifadəsi qısadır!'%(i))
    else:
        print('%s ifadəsi listəyə əlavə olundu'%(i))
        for x in list+[i]:
            if x:
                print(x)
```

```
write something:python
python ifadəsi listəyə əlavə olundu
python
write something:er
er ifadəsi qısadır!
write something:235347566789
235347566789 ifadəsi listəyə əlavə olundu
235347566789
write something:123456
123456 ifadəsi listəyə əlavə olundu
```

```
123456
write something:
```

del metodu

```
>>> list
['python', 'elsever']
>>> del list[0]
>>> list
['elsever']
>>>
```

əgər yaratdığınız list qarşısına del metodunu yazsanız listiniz tamamilə silinəcək.

```
>>> del list
>>> list
<class 'list'>
>>>
```

Biz ilk əvvəl listimizdəki ifadəni dəyişdirdik.Bunun başqa yolları metodlardan keçir.Gəlin listin ala bildiyi metodlara baxaq.Python komanda sətrindən

List daxilində operatorlar

```
>>> list=[i for i in range(100)]
>>> list
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78,
79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
98, 99]
>>>
```

range metodu 0-dan 100-ə qədər rəqəmləri listimizə daxil etdi

və ya

```
>>> list=[]
>>> for i in range(100):
	print(list+[i])
```

for operatoru

```
>>> list=[i for i in range(50)if i%2==1]
>>> list
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41,
43, 45, 47, 49]
>>>
```

range() 0-dan 50-ə qədər rəqəmləri yaradır,daha sonra bu rəqəmlərin içində olan tək ədədləri listə daxil edərək çap edir.eyni qayda ilə ==0 yazsanız cüt rəqəmləri alarsınız

və ya

```
>>> list=[]
>>> for i in range(50):
	if i%2==0:
		list+=[i]


>>> list
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40,
42, 44, 46, 48]
>>>
```

Son kodlarımızı istifadə edərkən hər zaman ilk başda bir boş list (list=[]) yaratmalısınız əks halda xəta alacaqsınız

```
>>> del list
>>> for i in range(50):
	if i%2==1:
		list+=[i]


Traceback (most recent call last):
  File "<pyshell#175>", line 3, in <module>
    list+=[i]
TypeError: unsupported operand type(s) for +=: 'type' and 'list'
>>>
```

Ifadələri qruplaşdırmaq

```
>>> list=[[1,2,3,'java'],[34,56,1,'Eduard'],[12,2]]
>>> list
[[1, 2, 3, 'java'], [34, 56, 1, 'Eduard'], [12, 2]]
>>> list[0]
[1, 2, 3, 'java']
>>> list[-1]
[12, 2]
>>> list[1]
[34, 56, 1, 'Eduard']
>>> len(list)
3
>>>
```

List daxilindəki ifadələrin sayı toplam 3-sayda oldu

0 - [1, 2, 3, 'java']
1 - [34, 56, 1, 'Eduard']
2 - [12,2]

List metodları

python komanda sətrindən

```
>>> list=[i for i in dir(list) if '_' not in i]
```

```
>>> list
['append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```

və ya

```
>>> for i in dir(list):
	if '_' not in i:
		print(i)
```

append() metodu

Dilimizə əlavə etmək kimi tərcümə olunur.listlərə ifadə əlavə etmək vəzifəsini daşıyır.Bu metod vasitəsilə yalnız bir ədəd ifadə daxil edə bilərik.Metod ifadəni list daxilində sonuncu sıraya əlavə edir.

```
>>> list=[]
>>> list.append(6)
>>> list
[6]
>>> list.append('Koala')
>>> list
[6, 'Koala']
>>> list.append('Jaguar')
>>> list
[6, 'Koala', 'Jaguar']
>>>
```

əgər birdən çox sayda ifadə əlavə etmək istəsək o zaman for() operatorundan istifadə edəcəyik.

```
>>> for i in ['tiger','wolf','eagle']:
```

```
    list.append(i)


>>> list
[6, 'Koala', 'Jaguar', 'tiger', 'wolf', 'eagle']
>>>
```

bundan öncəki bəhsdə keçdiyimiz müsbət(+) işarəsinin köməyilədə olur.

```
>>> list+=['sniper','deltatest','sololearn']
>>> list
[6, 'Koala', 'Jaguar', 'tiger', 'wolf', 'eagle', 'sniper', 'deltatest', 'sololearn']
>>>
```

clear()   metodu

Metod,list daxilindəki ifadələri silir.Dilimizə silmək kimi tərcümə olunur.
Metod list daxilindəki bütün ifadələri silir.

```
>>> list
[1, 23]
>>> list.clear()
>>> list
[]
>>>
```

copy() metodu

metod iki listi birini-digərinin üzərinə yapışdırır.

```
>>> list=[]
>>> list
[]
```

```
>>> list.append(1)
>>> list.append(23)
>>> list+['python','java','jython']
[1, 23, 'python', 'java', 'jython']
>>> list1=[]
>>> list1+[34,6,'rubby','alma','armud']
[34, 6, 'rubby', 'alma', 'armud']
>>> list1
[]
>>> list1=list.copy()
>>> list1
[1, 23]
>>> list
[1, 23]
>>> list1
[1, 23]
>>>
```

count()   metodu

Metod ifadə daxilində hərhansısa bir söz və ya rəqəmdən neçə dənə olduğunu təyin edir

```
>>> list
[1, 12, 45, 'wifi']
>>> list.append('pylab')
>>> list.append(1)
>>> list
[1, 12, 45, 'wifi', 'pylab', 1]
>>> list.count(1)
2
>>> list.append('wifi')
>>> list
[1, 12, 45, 'wifi', 'pylab', 1, 'wifi']
>>> list.count('wifi')
2
>>>
```

extend() metodu

Dilimizə genişlənmək kimi tərcümə olunur.metod iki list daxilindəki ifadələri toplayaraq bir tək list halına gətirir.

```
>>> list=[]
>>> list.append(12)
>>> list.append(4)
>>> list.append('function')
>>> list1=[]
>>> list1.append(16)
>>> list1.append('built')
>>> list1
[16, 'built']
>>> list.extend(list1)
>>> list
[12, 4, 'function', 16, 'built']
>>>
```

insert() metodu

Metod dilimizə yerləşdirmək kimi tərcümə olunur.append () metodundan fərqi,metod ifadəni göstərdiyimiz sıraya rahatlıqla yerləşdirməsidir.Metod iki argument alır

```
>>> list=[]
>>> list.append('python')
>>> list
['python']
>>> list.append('java')
>>> list
['python', 'java']
>>>
```

Gördüyümüz kimi append() metodundan hər dəfə istifadə etdikdə ifadə sona yerləşdirilir.İndi də insert() metoduna baxaq

```
>>> list.insert(0,'rubby')
>>> list
```

```
['rubby', 'python', 'java']
>>> list.insert(1,34)
>>> list
['rubby', 34, 'python', 'java']
>>>
```

sıra nömrələrin bildirməklə ifadələrimizi yerləşdirdik

```
>>> for i in [12,23,'c++']:
	list.insert(0,i)



>>> list
['c++', 23, 12, 'rubby', 34, 'python', 'java']
>>> len(list)
7
```

remove() metodu

Metod dilimizə kənarlaşdırmaq kimi tərcümə olunur.Metod ifadəni list daxilindən kənarlaşdırır.Yalnız bir argument alır.

```
>>> list.remove('c++')
>>> list
[23, 12, 'rubby', 34, 'python', 'java']
>>> list.remove(23)
>>> list
[12, 'rubby', 34, 'python', 'java']
>>>
```

reverse() metodu

Metod list ifadələrini tərs çevirir.

```
>>> list
[12, 'rubby', 34, 'python', 'java']
>>> list.reverse()
```

```
>>> list
['java', 'python', 34, 'rubby', 12]
>>>
```

pop() metodu

remove() metodu ilə listdən ifadə sildikdə,silinən ifadə ekrana çap olunmur.Amma pop metodu vasitəsilə sildiyimiz ifadə ekrana çap olunur.Bir argument alır

```
>>> list
[12, 'rubby', 34, 'python', 'java']
>>> list.pop(1)
'rubby'
>>> list.pop(-3)
34
>>> list
[12, 'python', 'java']
>>>
```

sort() metodu

Metod list daxilindəki ifadələri sözləri-əlifba sırası ilə düzür,rəqəmləri isə minimum dəyərindən maksimum dəyərə -kiçikdən böyüyə doğru düzür.

```
>>> list
['python', 'java', 'elitar', 'Guido Van Rossum']
>>> list.sort()
>>> list
['Guido Van Rossum', 'elitar', 'java', 'python']
>>>
>>> list=[]
>>> list.append(1)
>>> list.append(12)
>>> list.append(45)
>>> list
```

```
[1, 12, 45]
>>> type(list)
<class 'list'>
>>> max(list)
45
>>> min(list)
1
>>> list.sort()
>>> list
[1, 12, 45]
>>>
```

Əgər listdə qarışıq ifadələr yəni ədədlər və sözlər varsa sort() metodundan istifadə edə bilməyəcəksiniz.

```
>>> list
[1, 12, 45]
>>> list.append('wifi')
>>> list
[1, 12, 45, 'wifi']
>>> list.sort()
Traceback (most recent call last):
  File "<pyshell#60>", line 1, in <module>
    list.sort()
TypeError: unorderable types: str() < int()
>>>
```

index() metodu

Metod, argumenti sorğuya çəkməklə neçənci sırada olduğunu təyin edir.

```
>>> list
[1, 12, 45, 'wifi']
>>> list.index('wifi')
3
```

```
>>> list.index(12)
1
>>>
```

Tupllar.(tuple)

Tuplları həm mötərizə daxilində həmdə mötərizəsiz yaratmaq olur.Tupl listlərdən fərqlənir.Biz list metodları vasitəsilə,eləcədə digər metodlarla list daxilinə integer string cins ifadələri əlavə edib daha sonra dəyişiklik edə bilirdiksə,bunlar tupllar üçün keçərli deyil.

```
>>> tuple='NASA','Dropbox'
>>> type(tuple)
<class 'tuple'>
>>> tuple=()
>>> tuple
()
>>> type(tuple)
<class 'tuple'>
>>> i='NASA',23,'Elitar'
>>> i
('NASA', 23, 'Elitar')
>>> type(i)
<class 'tuple'>
>>> i=('Linux','OS7','Nt','Android')
>>> i
('Linux', 'OS7', 'Nt', 'Android')
>>> type(i)
<class 'tuple'>
>>>
```

Yuxarıdakı kodlarımızda fərqli üsullarla yəni həm mötərizə daxilində həmdə mötərizəsiz tupl yaratdıq və daha sonra tipini soruşduq,python

bizə tuple olduğunu bildirdi

tək ifadəli bir tupl yaratmaq üçün

```
>>> x=('Mayami',)
>>> x
('Mayami',)
>>>
```

və ifadədən sonra vergül qoymağı unutmuruq.

Listlərdə olduğu kimi tupllarda da qapalı mötərizədən istifadə edərək ifadə daxilində olanları ekrana çap edə bilərsiniz.

```
>>> i
('Linux', 'OS7', 'Nt', 'Android')
>>> type(i)
<class 'tuple'>
>>> i[0]
'Linux'
>>> i[1]
'OS7'
>>> i[-3]
'OS7'
>>> i[3]
'Android'
>>>
```

Yuxarıda qeyd etdiyimiz kimi tupllara list metodlarından istifadə edərək ifadə əlavə edə bilməzsiniz,bunun üçün str cinslərdə istifadə etdiyimiz variantlar keçərlidir

```
>>> i+('c++',)
('Linux', 'OS7', 'Nt', 'Android', 'c++')
>>> i+('driver','Nagios','Nginx')
('Linux', 'OS7', 'Nt', 'Android', 'driver', 'Nagios', 'Nginx')
```

Yuxarıdakı üsulla ifadə əlavə etmək müvəqqəti olduğundan davamlı istifadə olunan üsul deyil.Bunun üçün aşağıda yazacağım üsula nəzər salaq.

```
>>> i=i+('driver','Nagios','Nginx')
>>> i
('Linux', 'OS7', 'Nt', 'Android', 'driver', 'Nagios', 'Nginx')
>>>
```

təkrar komanda sətrindən I-ni çağırdıqda son əlavələrimiz göründü

Tupl metodları

Komanda sətrindən

```
>>> dir(tuple)
```

və ya

```
>>> tuple=(i for i in dir(tuple)if '_' not in i)
>>> tuple
<generator object <genexpr> at 0x7fcf64ca6200>
```

Yuxarıda tupl metodlarını görmək istədik amma ekrana çap olunmadı.Ekranda olan ifadə metodların varlığına dair bildirişdir.Biz metodları əyani görmək üçün

```
>>> print(*tuple)
count index
>>>
```

yazaraq əldə edə bilərik.
Deməli iki istifadə edə biləcəyimiz metod var.

count()   metodu   (tupllar üçün)

listlərdə olduğu kimi tupllarda da bir ifadənin tupl daxilində nəqədər olduğunu göstərir.

```
>>> tuple=()
>>> tuple+=('tupl','list','def','if','for')
>>> tuple
('tupl', 'list', 'def', 'if', 'for')
>>> tuple+=('de','2','if')
>>> tuple
('tupl', 'list', 'def', 'if', 'for', 'de', '2', 'if')
>>> tuple.count('def')
1
>>> tuple.count('if')
2
>>>
```

Və if -ifadəsinin iki dənə olduğu,def ifadəsindən birdənə olduğunu göstərdi

index()   metodu(tupllar üçün)

Listlərdə olduğu kimi tupllarda da index() metodu ifadənin neçənci sırada dayandığını göstərir.

```
>>> tuple
('tupl', 'list', 'def', 'if', 'for', 'de', '2', 'if')
>>> tuple.index('list')
1
>>> tuple.index('if')
3
>>>
```

Dictionary (lüğət)

Dilimizə lüğət kimi tərcümə olunur.List və tupllar kimi bu bəhsin də imkanları genişdir.
Bir boş dict yaradaq

```
>>> i={}
>>> type(i)
<class 'dict'>
```

Dict-lərdə cəm mötərizədən istifadə olunur.dict,daxilində iki növ ifadə daşıyır,biri values-dəyər,digəri keys-açar sözlər.Biri digərindən asılıdır,digər sözün açıqlamasıdır,aidliyidir.
Bunu kodlarımızda görək

```
>>> dict_i={'server':'db.python.org','database':'sql'}
>>> type(dict_i)
<class 'dict'>
>>> dict_i
{'server': 'db.python.org', 'database': 'sql'}
>>>
```

Yuxarıdakı ifadələrimizdə keys -server və database,values- db.python.org və sql dir.Yəni açar və dəyərlər

```
>>> dict_i.keys()
dict_keys(['server', 'database'])
>>> dict_i.values()
dict_values(['db.python.org', 'sql'])
```

Əgər dəyər (values) ifadəni sorğuya çəksək xəta alacağıq

```
>>> dict_i['sql']
Traceback (most recent call last):
  File "<pyshell#163>", line 1, in <module>
    dict_i['sql']
KeyError: 'sql'
```

```
>>>
```

amma açar(keys) sözləri rahatlıqla soruşa bilərik

```
>>> dict_i['server']
'db.python.org'
>>> dict_i['database']
'sql'
```

dəyər(values) ifadələri dəyişdirmək

```
>>> dict_i
{'server': 'mysql', 'database': 'sql'}
>>> dict_i['server']='db.github.com'
>>> dict_i
{'server': 'db.github.com', 'database': 'sql'}
>>> dict_i['database']='mysql'
>>> dict_i
{'server': 'db.github.com', 'database': 'mysql'}
>>>
```

açar və dəyər sözlər əlavə etmək

```
>>> dict_i
{'server': 'db.github.com', 'database': 'mysql'}
>>> dict_i['password']='hexdic32'
>>> dict_i
{'server': 'db.github.com', 'database': 'mysql', 'password': 'hexdic32'}
>>>
```

```
 keys         values            keys      values      keys        values
  |             |                 |          |          |            |
{'server': 'db.github.com', 'database': 'mysql', 'password': 'hexdic32'}
```

dəyərlərə birdən çox ifadələr əlavə etmək

```
>>>
dict_i['server']='db.invention.org','db.aliexpress.com','db.canonical.gov'
>>> dict_i
{'server': ('db.invention.org', 'db.aliexpress.com', 'db.canonical.gov'),
'database': 'mysql', 'password': 'hexdic32'}
>>> dict_i.values()
dict_values([('db.invention.org', 'db.aliexpress.com', 'db.canonical.gov'),
'mysql', 'hexdic32'])
>>> dict_i
{'server': ('db.invention.org', 'db.aliexpress.com', 'db.canonical.gov'),
'database': 'mysql', 'password': 'hexdic32'}
>>> dict_i.keys()
dict_keys(['server', 'database', 'password'])
>>>
```

Yuxarıdakı kodlarımızda

```
'server': ('db.invention.org', 'db.aliexpress.com', 'db.canonical.gov')
'database': 'mysql'
'password': 'hexdic32'
```

uyğun gələcək

lüğətlərin(dictionary) metodları

```
>>> dir(dict)
```

və ya

```
>>> for i in dir(dict):
        if '_'not in i:
                print(i)
```

```
clear
copy
fromkeys
get
items
keys
pop
popitem
setdefault
update
values
>>>
```

Mövzumuza girişdə keys və values metodlarından istifadə etmişdik.Gəlin bir də təkrar edək.
keys-açar
values-dəyər sözlər

```
>>> dict={}
>>> dict['port']=25,45,80,467,587,22
>>> dict
{'port': (25, 45, 80, 467, 587, 22)}
>>> dict['url']='python.org','tqdk.gov.az'
>>> dict
{'url': ('python.org', 'tqdk.gov.az'), 'port': (25, 45, 80, 467, 587, 22)}
>>> dict.keys()
dict_keys(['url', 'port'])
>>> dict.values()
dict_values([('python.org', 'tqdk.gov.az'), (25, 45, 80, 467, 587, 22)])
>>>
```

items()   metodu

Metod lüğətdən həm açar,həm də dəyər sözləri çap edirik.python3

```
>>> dict.items()
```

```
dict_items([('url', ('python.org', 'tqdk.gov.az')), ('port', (25, 45, 80, 467,
587, 22))])
>>>
```

get()   metodu

Metod iki argument alır.Misallara baxaq

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
while True:
    dict={'url': ('python.org', 'tqdk.gov.az'),
            'port': (25, 45, 80, 467, 587, 22)}
    i=input('bazadan axtarış üçün ifadəni yazın:')
    if i not in dict:
        print('ifadəniz təəssüfki bazada yoxdur!')

    else:
        print(dict.get(i,'axtardığınız söz bazada var:'))
```

```
bazadan axtarış üçün ifadəni yazın:url
('python.org', 'tqdk.gov.az')
bazadan axtarış üçün ifadəni yazın:rt
ifadəniz təəssüfki bazada yoxdur!
bazadan axtarış üçün ifadəni yazın:port
(25, 45, 80, 467, 587, 22)
bazadan axtarış üçün ifadəni yazın:
```

clear()   metodu

Metod lüğət daxilindəki açar və dəyər sözləri tamamilə silir.Dilimizə silmək kimi tərcümə olunur.

```
>>> dict
{'url': ('python.org', 'tqdk.gov.az'), 'port': (25, 45, 80, 467, 587, 22)}
>>> dict.clear()
>>> dict
{}
>>>
```

copy() metodu

Metod dilimizə yapışdırmaq,köçürmək kimi tərcümə olunur.Misallara baxaq

```
>>> dict
{'url': ('python.org', 'tqdk.gov.az'), 'port': (25, 45, 80, 467, 587, 22)}
>>> dict_new=dict
>>> dict
{'url': ('python.org', 'tqdk.gov.az'), 'port': (25, 45, 80, 467, 587, 22)}
>>> dict_new
{'url': ('python.org', 'tqdk.gov.az'), 'port': (25, 45, 80, 467, 587, 22)}
>>>
```

dict_new=dict ifadəsini yazmaqla bir lüğətimizdən daha birini yaratdıq

fromkeys() metodu

```
>>> tuple='python','java','rubby'
>>> dict=dict.fromkeys(tuple,'programlama dili')
>>> dict
{'rubby': 'programlama dili', 'python': 'programlama dili', 'java':
'programlama dili'}
>>>
```

metod daxilindəki ifadə sadəcə values-dəyər söz kimi hər birinə əlavə olundu.
Bunu listlərlə də edə bilərik.Metod bu funksiyasını list və tupllarla yerinə yetirir.

```
>>> list=['Linkoln','Corc Bush','Obama']
>>> list
['Linkoln', 'Corc Bush', 'Obama']
>>> dict=dict.fromkeys(list,'president')
>>> dict
{'Corc Bush': 'president', 'Obama': 'president', 'Linkoln': 'president'}
>>>
```

pop() metodu

Metod listlərdən fərqli olaraq lüğətlərdə bir argument alaraq(açar sözü) silir

```
>>> dict
{'Corc Bush': 'president', 'Obama': 'president', 'Linkoln': 'president'}
>>> dict.pop('Corc Bush')
'president'
>>> dict
{'Obama': 'president', 'Linkoln': 'president'}
>>>
```

Listlərin bir xassəsi isə əgər açar söz və ya dəyər söz silinibsə o zaman o ifadə tamamilə silinəcək.Son kodlarımızda da biz dəyər(president) sözü silməklə açar(Corc Bush) sözü də sildik.

popitem() metodu

Metod,ifadə daxilində (dict) təsüdüf açar və dəyər sözü silir.Bu səbəbdən ala biləcəyi argument yoxdur.Metod demək olarki əksər hallarda ilk açar-dəyər ifadəsini silir.

```
>>> i={'Azərbaycan': 'Bakı' ,'İçərişəhər': 'Qalaqapısı' ,'Səməd Vurğun küçəsi': 'Fəvvarələr meydanı'}
>>> i
{'Azərbaycan': 'Bakı', 'İçərişəhər': 'Qalaqapısı', 'Səməd Vurğun küçəsi': 'Fəvvarələr meydanı'}
```

```
>>> i.popitem()
('Azərbaycan', 'Bakı')
>>> i
{'İçərişəhər': 'Qalaqapısı', 'Səməd Vurğun küçəsi': 'Fəvvarələr meydanı'}
>>> i.popitem()
('İçərişəhər', 'Qalaqapısı')
>>> i
{'Səməd Vurğun küçəsi': 'Fəvvarələr meydanı'}
>>>
```

setdefault()   metodu

Metod,ifadə daxilində yeni açar və dəyər söz yaratmaq imkanına malikdir.

```
>>> i
{'Səməd Vurğun küçəsi': 'Fəvvarələr meydanı'}
>>> i.setdefault('USA','Vashinqton')
'Vashinqton'
>>> i
{'USA': 'Vashinqton', 'Səməd Vurğun küçəsi': 'Fəvvarələr meydanı'}
```

Əgər bir açar sözə birdən artıq dəyər söz yazmaq istəsək,o zaman dəyər sözləri mötərizə daxilində yazmalıyıq.

```
>>> i.setdefault('America',('ABŞ','Vaşinqton','Obama'))
('ABŞ', 'Vaşinqton', 'Obama')
>>> i
{'America': ('ABŞ', 'Vaşinqton', 'Obama'), 'USA': 'Vashinqton', 'Səməd
Vurğun küçəsi': 'Fəvvarələr meydanı'}
```

uptade()   metodu

Metod, copy() metoduna bənzəyir.Fərq ondadırki bu metod iki dict üçün ayrı-ayrılıqda yeni dict yaratmır.Bir növ linux sistemlərinin update

olunması kimidir.Biz bir dict yaradırıq daha sonra köhnə dict daxilinə yeni dict ifadələrlərini əlavə edirik və ya tərsi olaraq.Misallardan daha aydın olacaq

```
>>> i
{'America': ('ABŞ', 'Vaşinqton', 'Obama'), 'USA': 'Vashinqton', 'Səməd Vurğun küçəsi': 'Fəvvarələr meydanı'}
>>> new_i={'Albania':'Tirana','South Africa':'Keyptaun','Chili':'Snatiaqo'}
>>> i.update(new_i)
>>> i
{'South Africa': 'Keyptaun', 'Albania': 'Tirana', 'Səməd Vurğun küçəsi': 'Fəvvarələr meydanı', 'Chili': 'Snatiaqo', 'America': ('ABŞ', 'Vaşinqton', 'Obama'), 'USA': 'Vashinqton'}
>>>
```

String,list,tuple və dict -dən başqa çoxluqlar adlanan növbəti bəhsimiz var.
Çoxluqlar da öz növbəsində iki funksiya ilə yaradılır.İlk funksiyası set,digəri isə frozenset adlanır.Fərqləri isə set funksiyası daxilində olan ifadələrə dəyişiklik etmək olur,amma frozenset funksiyasında bu mümkün deyil,çünki adından da aşkar olduğu kimi dondurulmuş qurma,quraşdırma kimi mənaları ifadə edir.set-funksiyasını biz dilimizə qurmaq kimi tərcümə edək.Yəni funksiya vasitəsilə yeni bir çoxluq qururuq,yaradırıq.İlk boş bir çoxluq yaradacağımıza nəzər yetirək

```
>>> i=set()
>>> type(i)
<class 'set'>
>>>
```

Tipini soruşduqda bizə set-olaraq çap etdi

və ya

```
>>> i=frozenset()
>>> type(i)
```

```
<class 'frozenset'>
>>>
```

```
>>> x=set('kolbasa','sosis','kabab')
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    x=set('kolbasa','sosis','kabab')
TypeError: set expected at most 1 arguments, got 3
>>>
```

Yuxarıda gördüyünüz kimi xəta aldıq,çünki biz birbaşa list,tupl,string və dict-dən istifadə edə bilmərik.İrəlidə bunlarla bağlı misallar yazacağıq.Və davam edək

```
>>> x=set(['kolbasa','sosis','kabab'])
>>> x
{'kolbasa', 'sosis', 'kabab'}
>>> type(x)
<class 'set'>
>>>

>>> i=set({'alma','armud','heyva'})
>>> i
{'alma', 'armud', 'heyva'}
>>> type(i)
<class 'set'>
>>>
```

çoxluq yaratdıq.

Eləcədə string,list,tupl və ya dict -i set funksiyası ilə çevirərək çoxluq yarada bilərik

```
>>> list=['guys','boys','letter']
>>> x=set(list)
>>> x
{'guys', 'letter', 'boys'}
```

```
>>> type(x)
<class 'set'>
>>>

>>> i='Azərbaycan Neft Akademiyası'
>>> x=set(i)
>>> x
{'ı', 'c', 'N', 'y', 'e', 'a', 'b', 'm', 'i', 'A', 'n', 'f', 's', 'r', 'ə', 'd', 'k', 't', 'z', ' '}
```

Yuxarıda başqa nəticə gözlədiyimiz halda stringlərdə, çoxluq tamam başda bir şey çap etdi.Qeyd edimki çoxluqlar ardıcıllıq tanımır,eləcədə eyni adları,hərfləri təkrar etmir.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
dict={'santexnika','texnoloq','Brayn','Nike','texnoloq','woodalien','santexn
ika'}
for i in dict:
    count=len(i)
    print('{} uzunluğu {}'.format(i,count))


woodalien uzunluğu 9
texnoloq uzunluğu 8
santexnika uzunluğu 10
Nike uzunluğu 4
Brayn uzunluğu 5
>>>
```

```
>>> dict={'selfie':'photo','GIF':'videocapt'}
>>> x=set(dict)
>>> x
{'GIF', 'selfie'}
```

Qeyd edimki biz ifadələri çoxluq içində daxil edərək yaratdıq.Çoxluq yaradarkən list,tupl və lüğətlərdəki mötərizələrdən istifadə edirik.Amma bir məsələ varki boş bir çoxluq yaratmaq istəsək bu mümkün olmayacaq.

```
>>> set={}
>>> set
{}
>>> type(set)
<class 'dict'>
```

Biz ,heç bir halda boş bir çoxluq yarada bilmədik.Yuxarıda da qeyd etdiyim kimi bunu çevirmə yolu ilə və ya birbaşa set metodundan istifadə edərək.

Çoxluqların metodları

```
>>> for i in dir(set):
	if '__' not in i:
		print(i)



add
clear
copy
difference
difference_update
discard
intersection
intersection_update
isdisjoint
issubset
issuperset
pop
remove
symmetric_difference
symmetric_difference_update
union
update
>>>
```

Yuxarıda əldə etdiyimiz metodları tək-tək öyrənək
Biz hal-hazırda dəyişdirilə bilən (set) funksiyasını öyrəndiyimiz üçün çoxluqlar bu metodlar vasitəsilə dəyişdirilə bilinir.

add()   metodu

Metod dilimizə əlavə etmək kimi tərcümə olunur.Metod vasitəsilə ifadələri çoxluq daxilinə əlavə edəcəyik.

```
#!/usr/bin/python
dict={'CNN','Meydantv','Aljazeria'}
x=set(dict)
x.add('BBC')
print (x)

set(['CNN', 'BBC', 'Meydantv', 'Aljazeria'])
```

Bu metod vasitəsilə birbaşa yalnış tək bir ifadə əlavə edə bilirik.Əgər çox sayda ifadələr əlavə etmək istəsək for operatorundan istifadə edəcəyik

```
#!/usr/bin/python
v=set(['BBC','Chinetv','Dehlitv'])
x=['ChineRadio','Elephantv','Fizella']
for i in x:
        v.add(i)
        print(v)


set(['Chinetv', 'ChineRadio', 'Dehlitv', 'BBC'])
set(['Chinetv', 'ChineRadio', 'Elephantv', 'Dehlitv', 'BBC'])
set(['Elephantv', 'ChineRadio', 'BBC', 'Fizella', 'Chinetv', 'Dehlitv'])
```

clear()   metodu

Metod dilimizə silmək,təmizləmək kimi tərcümə olunur.Metod çoxluqların içərisini silir,yəni daxilində olan bütün ifadələri silir.

```
>>> x=set(['Dan Brown'])
>>> x
{'Dan Brown'}
>>> type(x)
<class 'set'>
>>> x.clear()
>>> x
set()
>>>
```

İlk əvvəl ifadəsi 'Dan Brown' olan çoxluq yaratdıq,daha sonra tipini soruşaraq çoxluq olduğuna əmin olduq və son kodlarımızda clear() metodundan istifadə edərək çoxluq ifadəsini sildik

copy()   metodu

Metod dilimizə yapışdırmaq kimi tərcümə olunur.Metod hazır çoxluğu digərinə yapışdırır.

```
>>> x=set(['Bayer','Celtik','Neftçi'])
>>> x
{'Celtik', 'Neftçi', 'Bayer'}
>>> v=x.copy()
>>> v
{'Celtik', 'Neftçi', 'Bayer'}
>>> x
{'Celtik', 'Neftçi', 'Bayer'}
>>> type(v)
```

```
<class 'set'>
>>> type(x)
<class 'set'>
>>>
```

Amma təəssüflər olsunki lüğətlərdə olduğu kimi çoxluqlarda da bəzən ardıcıllıq pozulur.Bu daha çox özünü string-də özünü biruzə verir.Misallara baxaq

```
>>> x=set('dragon era')
>>> i=x.copy()
>>> i
{' ', 'a', 'g', 'r', 'o', 'd', 'e', 'n'}
>>>
```

Yuxarıdakı nəticədən göründüyü kimi ardıcıllıq pozulur.

Copy metodunu = -işarəsi ilə də əvəz edərək yeni çoxluq yarada bilərik

```
>>> i=x
>>> i
{' ', 'n', 'g', 'e', 'o', 'r', 'd', 'a'}
>>> x
{' ', 'n', 'g', 'e', 'o', 'r', 'd', 'a'}
>>>
```

difference()   metodu

Metod dilimizə fərq,müxtəliflik kimi tərcümə olunur.Metod iki ayrı təyin olunmuş çoxluq ifadələrini qarşılaşdıraraq,ikisi arasındakı fərqli ifadələri təyin edib, çap edir.

```
>>> x=set(['v','t','a','i','n'])
>>> v=set(['v','b','n','t','f'])
```

```
>>> x.difference(v)
{'i', 'a'}
>>>
```

Deməli x-çoxluğunda olan i və a hərfləri,v-çoxluğunda yoxdur.
İndidə tam tərsin yazaraq

```
>>> v.difference(x)
{'f', 'b'}
>>>
```

Son nəticəmizdə, v-çoxluqunda olan f və b x-çoxluğunda yoxdur.

Və ya mənfi(-) işarəsindən istifadə edərək yuxarıdakı nəticələri əldə edə bilərik.

```
>>> x-v
{'i', 'a'}
>>> v-x
{'f', 'b'}
>>>
```

difference_update()   metodu

Metod sizdə qarışıqlıq yarada bilər.Metodumuz iki çoxluq arasında ortaq qiymətləri tapıb,metodu tətbiq etdiyimiz çoxluq ifadəsini ortaq qiymətlər deyil,qeyri-ortaq olan ifadələri yerləşdirir.Misallardan daha aydın olacaq

```
>>> x=set([1,2,3])
>>> i=set([1,3,5])
>>> x.difference_update(i)
>>> x
{2}
>>> i
{1, 3, 5}
>>>
```

İlk əvvəl x-çoxluğunu ifadələrlə birgə tətbiq etdik,dahan sonra I-çoxluğuna saylar verərək tətbiq etdik.Ardıcıllıqla metodumuzdan istifadə edərək,metodu x-çoxluğuna tətbiq etdik.Aldığımız nəticə isə 2-ədədi oldu.Belə aydın olurki 1 və 3 ədədləri I-çoxluğunda da olduğu üçün x-çoxluğundan silindi və yerdə qalan 2- ədədi x-çoxluğumuza tətbiq oldu.Başqa misallara baxaq

```
>>> x=set(['Berlin','Santyago','Baki'])
>>> i=set(['Santyago','Aljazera','Tunisia'])
>>> x.difference_update(i)
>>> x
{'Berlin', 'Baki'}
>>>
```

Son kodlarımızda ifadələr x-çoxluğunda olan Santiago,i-çoxluğunda da olduğuna görə tətbiq olunmada yalnız iki ifadə əlimizdə qaldı(Berlin və Baki)

discard()   metodu

Metod çoxluq daxilində olan ifadəni silməyimizə yardımçı olur.Amma metod vasitəsilə çoxluq daxilində olmayan ifadəni silməyə çalışsaq,heç bir xəta mesajı almayacağıq.

```
>>> x=set(['Berlin','Santyago','Baki'])
>>> x.discard('Berlin')
>>> x
{'Santyago', 'Baki'}
>>>

>>> x.discard('almera')
>>> x
{'Santyago', 'Baki'}
>>>
```

Yuxarıda qeyd etdiyimiz kimi,metod vasitəsilə almera ifadəsini silməyə çalışdıq amma xəta almadıq.İfadə daxilində bu söz yoxdur.

remove()   metodu

Metod discard() metodu kimi çoxluqdan bir ifadəni silir.

```
>>> x.remove('Baki')
>>> x
{'Santyago'}
>>> x.remove('Almaniya')
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    x.remove('Almaniya')
KeyError: 'Almaniya'
>>> x
{'Santyago'}
>>>
```

Son nəticəmizdə xəta aldıq.Elə discard() metodu ilə remove() metodun arasında fərq də bundadır.discard() metodunda,olmayan ifadəni silmə əmri verdikdə xəta mesajı almırıq,amma bu metodda xəta mesajı alırıq.

Intersection()   metodu

Metod dilimizə kəsişmə,ortaq kimi tərcümə olunur.Pythonda da öz funksiyasını tərcüməsi qədər yerinə yetirir.İki çoxluq arasında ortaq ifadələri çap edir.

```
>>> x=set({'flango','barbar','vikings'})
```

```
>>> v=set({'flango','vikings','largo'})
>>> x.intersection(v)
{'flango', 'vikings'}
>>> x=set({1,2,3,4,5})
>>> v=set({1,3,5,7,9})
>>> x.intersection(v)
{1, 3, 5}
>>>
```

kodlarımızda x və v çoxluqları yaratdıq,daha sonra metodumuzdan istifadə edərək hər iki çoxluq arasındakı ortaq ifadələri tapdıq.

intersection_update()   metodu

Metod iki çoxluq arasındakı ortaq fərqi taparaq,metodu yönləndirdiyimiz çoxluğa həmin ortaq fərqləri tətbiq edir.Misallardan fikrimiz aydın olacaq

```
>>> x
{1, 2, 3, 4, 5}
>>> v
{9, 1, 3, 5, 7}
>>> x.intersection_update(v)
>>> x
{1, 3, 5}
>>> v
{9, 1, 3, 5, 7}
>>> v.intersection(x)
{1, 3, 5}
>>> v
{9, 1, 3, 5, 7}
>>>
```

isdisjoint() metodu

Metod ,iki çoxluq arasında ortaq ifadələrin olub-olmadığını sorğuya çəkir.Əgər varsa False(inkar),yoxdursa True(təsdiq) ifadəsini çap edir. Yəni iki çoxluq arasında ortaq fərqlər varsa False,yoxdursa True verir.

```
>>> x=set({'a','b','c','d'})
>>> v=set({'e','f','g','h'})
>>> x.isdisjoint(v)
True
>>>
```

Gördüyümüz kimi hər iki çoxluq arasında eyni ifadələr olmadığından biz True ifadəsini aldıq.
Və ya

```
>>> x=set({'a','b','c','d','e'})
>>> x.isdisjoint(v)
False
>>> x
{'c', 'e', 'd', 'a', 'b'}
>>> v
{'g', 'e', 'h', 'f'}
>>>
```

x-çoxluğuna e-hərfini əlavə edərək v-çoxluğu ilə ortaqlıq yaratdıq və metdumuz bizə false verdi yəni bu iki çoxluq arasında oxşar ifadə,ifadələr var.

issubset() metodu

Bu metodun digər metodlara oxşarlığı var.Metod bir çoxluğun içində olan bütün ifadələri digər çoxluq ifadələri ilə qarşılaşdırır.Əgər ifadələrin hamısı varsa,ekran çapında True yoxdursa False ifadələri verir.Misallara baxaq

```
>>> x=set({'a','b','c'})
>>> v=set({'a','b','c','d','e'})
>>> x.issubset(v)
True
>>>
```

True ifadəsini aldıq,çünki a,b və c hərfləri v-çoxluğunun içində var

və ya

```
>>> x
{'c', 'e', 'd', 'a', 'b'}
>>> v
{'g', 'e', 'h', 'f'}
>>> x.issubset(v)
False
```

Yalnız bir e-hərfinin olduğu üçün biz False aldıq

metoddan əlavə >,<,>= işarələrindən istifadə edərək təyin edə bilərik

```
>>> x>v
False
>>> x<v
True
>>>
```

issuperset()   metodu

Metod iki çoxluq arasında ifadələrin çox olduğunu fərqini ortaya çıxarır.Daha dəqiq izah etsək,böyük və kiçik işarələrindən istifadə edərək hansının ifadələri çox olduğunu təyin edə bilərik.Və ya digər yolu bir-birini absorb(udmaq,canına çəkmək)edərək təyin edə bilərik.

İlk öncə çoxluq yaradaq

```
>>> x=set({'c', 'e', 'd', 'a', 'b'})
```

daha birini

```
>>> v=set({'c', 'a', 'b'})
```

Yaratdığımız bu iki çoxluqdan x-çoxluğunun ifadələri sayca çoxdur. böyük(>) ,kiçik və ya bərabərlik işarələrindən istifadə edərək təyin edək

```
>>> x>v
True
>>> x<v
False
>>> x>=v
True
>>> x<=v
False
>>>
```

Gördüyümüz kimi iki çoxluqdan hansının böyük(ifadələri sayca çox olan) olduğunu təyin etdik.Başqa bir yolu metodumuzdan isifadə edərək təyin edək

```
>>> x=set({'c', 'e', 'd', 'a', 'b'})
>>> v=set({'c', 'a', 'b'})
>>> x.issuperset(v)
True

>>> v.issuperset(x)
False
```

union() metodu

Metod iki çoxluq daxilində olan ifadələri toplayaraq bir yerdə çap edir.Və ya düz slash | işarəsindən istifadə edərək yerinə yetirə bilərik.

```
>>> x=set((1,2,3,4))
>>> v=set((5,6,7,8,9,90))
>>> x.union(v)
{1, 2, 3, 4, 5, 6, 7, 8, 9, 90}
>>>
```

```
>>> x
{1, 2, 3, 4}
>>> v
{5, 6, 7, 8, 9, 90}
>>> x|v
{1, 2, 3, 4, 5, 6, 7, 8, 9, 90}
>>>
```

update() metodu

Metod add() metoduna bənzəyir,yəni çoxluq daxilinə bir neçə sayda ifadə əlavə etməyimizə yardımçı olur.add() metodunda biz for operatorundan istifadə edərək çox sayda ifadə əlavə edirdiksə,update() metodunda for operatoruna ehtiyac yoxdur.Misallara baxaq

```
>>> i=(('php','c++','java SE','java EE','rubby'))
```

```
>>> i=set(('php','c++','java SE','java EE','rubby'))
>>> v=[1,2,3]
>>> i.update(v)
>>> i
{1, 'php', 2, 3, 'java SE', 'c++', 'rubby', 'java EE'}
>>>
```

```
>>> i={'r','t','v'}
>>> v=set(('php','c++','java SE','java EE','rubby'))
>>> v.update(i)
>>> v
{'php', 'java SE', 'v', 'c++', 'r', 'rubby', 'java EE', 't'}
>>> type(i)
<class 'set'>
>>> type(v)
<class 'set'>
>>>
```

symmetric_difference()   metodu

Metod iki çoxluq daxilində olan fərqli ifadələri təyin edir.Yəni iki çoxluq arsındakı fərqli ifadələri çap edirik.

```
>>> x=set((1,2,3,4))
>>> v=set((1,3,5,7))
>>> x.symmetric_difference(v)
{2, 4, 5, 7}
>>>
```

symmetric_difference_update()   metodu

Metod iki çoxluğu müqaisə edir digərində olan ifadələri metodu tətbiq etdiyimiz çoxluğa əlavə edirik.

```
>>> x=set((1,2,3,4))
>>> v=set((3,5,6,7))
>>> x.symmetric_difference_update(v)
>>> x
{1, 2, 4, 5, 6, 7}
>>> v
{3, 5, 6, 7}
>>>
```

pop()   metodu

Metod çoxluq daxilindən təsüdüf bir ifadəni silir.Biz bu tip metoda əvvəlki bəhslərimizdə öyrənmişdik və orda pop() metodu argument alırdı.Amma çoxluqlarda heç bir argument almayaraq təsadüf bir ifadəni silir

```
>>> i=set(('tango','whatsupp','vk','wamba','facebook'))
>>> type(i)
<class 'set'>
>>> i.pop()
'wamba'
>>> i
{'facebook', 'vk', 'tango', 'whatsupp'}
>>>
```

Bölümə başlamamışdan əvvəl çoxluqların iki funksiyası ilə yaradıldığını qeyd etmişdik.Biri set() digəri frozenset()

Qeyd etdiyimiz kimi frozentset() funksiyası ilə yaratdığımız çoxluqlara dəyişiklik edə bilmirik.dir(set) yazaraq funksiyanın metodlarına baxdığımız kimi,bu kodu frozenset() -ə tətbiq edərək ala biləcəyi metodlara baxa bilərik.

```
>>> for i in dir(frozenset):
	if '__' not in i:
		print (i)


copy
difference
intersection
isdisjoint
issubset
issuperset
symmetric_difference
union
>>>
```

Əldə etdiyimiz funksiyanın metodları arasında add() update() remove() görmədik.

```
>>> i=frozenset(('alma','armud','heyva','nar','üzüm'))
>>> type(i)
<class 'frozenset'>
>>> i
frozenset({'nar', 'üzüm', 'alma', 'armud', 'heyva'})
>>>
```

```
>>> i.add('nar')
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    i.add('nar')
AttributeError: 'frozenset' object has no attribute 'add'
>>>
```

Son kodlarımızda aldığımız xətaya görə frozenset() funksiyası ilə yaratdığımız çoxluqda add() metodundan istifadə edə bilmərik.Buna xəta almaqla bir daha əmin olduq.

```
>>> i=frozenset(('alma','armud','heyva','nar','üzüm'))
>>> type(i)
<class 'frozenset'>
>>> i
frozenset({'nar', 'üzüm', 'alma', 'armud', 'heyva'})

>>> x=i.copy()
>>> x
frozenset({'nar', 'üzüm', 'alma', 'armud', 'heyva'})
>>> i
frozenset({'nar', 'üzüm', 'alma', 'armud', 'heyva'})
```

Fayllar

Biz print() funksiyasını keçərkən,open() funksiyası haqqında danışmışdıq.open() funksiyası mətn fayllarını açmaqda bizə yardımçı olurdu.

```
>>> f=open('sahil.txt','a')
>>> f
<_io.TextIOWrapper name='sahil.txt' mode='a' encoding='UTF-8'>
```

Yuxarıdakı kodlarımızda sahil.txt adlı a-modunda bir mətn faylı açdıq və ya yaradacağımız mətn faylı üçün ünvan göstərərək yerinə yetirə bilərik.

```
>>> f=open('/home/user/marko_polo.rb','a')
>>> f
<_io.TextIOWrapper name='/home/user/marko_polo.rb' mode='a'
encoding='UTF-8'>
>>>
```

Əvvəlki bəhslərimizdə istiqametini qeyd etdiyimiz qovluqları hansı işarə ilə yazmağımızdan danışmışdıq.
windows -istifadəçiləri bu qeydə diqqətlə nəzər yetirməlidirlər.

```
>>> f=open('C:\user\python\marko_polo.rb','w')
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated \uXXXX escape
```

(rb-rubby programlama dilinə uyğun fayl tipidir.)

Və xəta aldıq,çünki \u -birləşməsini biz əvvəlki bəhslərimizdə keçmişik.Bu tip xətalardan qaçmaq üçün python bizə \\ -işarəsini və ya r -xüsusi hərfini təklif edir.

```
>>> f=open('C:\\user\python\marko_polo.rb','w')
>>> f
<_io.TextIOWrapper name='C:\\user\\python\\marko_polo.rb' mode='w' encoding='UTF-8'>
>>>
```

Və xəta almadan marko_polo.rb adlı fayl yaratdıq.

Fayl modları

Python-shell dən v və ya terminaldan pythonu çağıraraq;

>>> help(open) yazmaqla fayl modlarına,eləcədə onlardan istifadə qaydalarına rast gəlirik.hal-hazırda bizə aşağıdakı mövzu lazım olacaq.

```
Character Meaning
    --------- ---------------------------------------------------------------
'r'      open for reading (default)
'w'      open for writing, truncating the file first
'x'      open for exclusive creation, failing if the file already exists
'a'      open for writing, appending to the end of the file if it exists
'b'      binary mode
't'      text mode (default)
```

'+'   open a disk file for updating (reading and writing)
'U'   universal newlines mode (deprecated)

Qeyd edimki irəlidə fayl modlarına uyğun metodlara nəzər yetirəcəyik. Bəzi Fayl modlarına açıqlıq gətirək

r-modu

faylı oxuma modunda açar.Sisteminizdə mövcud olan faylı oxuma modunda açaraq,daxilindən ifadə oxuyur.Əgər sisteminizdə həmin adda fayl yoxdursa siz xəta mesajı alacaqsınız

```
>>> f=open('github.lst','r')
Traceback (most recent call last):
  File "<pyshell#136>", line 1, in <module>
    f=open('github.lst','r')
FileNotFoundError: [Errno 2] No such file or directory: 'github.lst'
>>>
```

Və xəta mesajında deyildiyi kimi bu adda fayl mövcud deyil.Qeyd edimki bu modda faylı göstərmək üçün digər yolu isə modu yazmamaqla da olar.Eyni qayda ilə

```
>>> f=open('github.lst')
Traceback (most recent call last):
  File "<pyshell#137>", line 1, in <module>
    f=open('github.lst')
FileNotFoundError: [Errno 2] No such file or directory: 'github.lst'
>>>
```

w -modu

fayla ifadə yazmaq üçün istifadə olunur.Bu modda olan fayl əgər sisteminizdə varsa,ifadələr silinəcəkdir.Əgər yoxdursa python bizə qeyd etdiyimiz ada uyğun bir fayl açaraq modunu da w- yaradacaq.

x- modu

faylı yazmaq modunda açır.Sisteminizdə əgər qeyd etdiyiniz adda fayl varsa python sizə xəta mesajı verəcəkdir.

```
>>> f=open('telbook.lst','x')
Traceback (most recent call last):
  File "<pyshell#138>", line 1, in <module>
    f=open('telbook.lst','x')
FileExistsError: [Errno 17] File exists: 'telbook.lst'
>>>
```

Sistemimizdə telbook.lst adındabir fayl olduğu üçün xəta mesajı aldıq. Modun w-modundan fərqi də elə bundadırki bu mod(x) birbaşa fayl daxilindəki ifadələri silmir,əvəzinə xəta mesajı verir.

Yuxarıdakı fayl tiplərindən istifadə edərək,həmin mətn fayllarımıza ifadələr yazaq.Bunun üçün bizə write() metodu lazım olacaq.

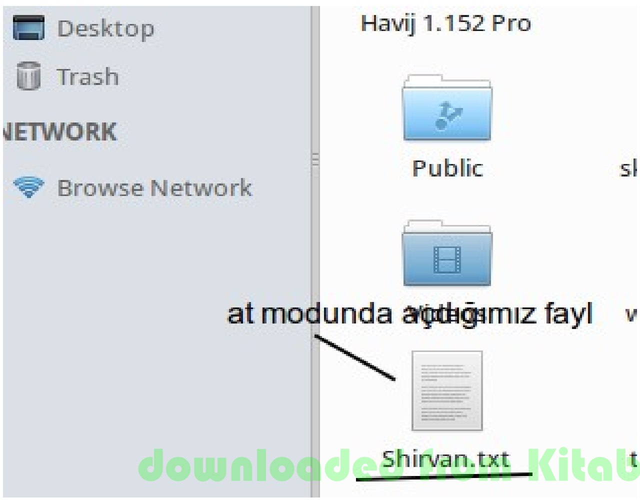
a-modu

Bir fayla ifadələr yazmaq hüququ verir.Sisteminizdə mövcud adda fayl varsa mod,fayl daxilindəki ifadələrə toxunmadan yeni ifadənizi fayla əlavə edəcək.Əgər qeyd etdiyiniz adda fayl yoxdursa python sizin üçün bu modda yeni bir fayl yaradacaq.

t modu

Digər modlarla brabər işlənə bilir.text faylları üçün istifadə olunur.Geniş yayılan mod deyildir

```
>>> f=open('Shirvan.txt','at')
>>> f.mode
'at'
>>>
```



+ modu

Bu mod tək işlənməyərək digər modlara əlavə olunaraq bir neçə funksiyanı yerinə yetirir.Yəni bir fayla yazma,oxuma hüququ verir.

a+ modu

Bu mod vasitəsilə faylı eyni anda oxumaq və yazmaq modunda açmaq olur.Əgər sisteminizdə fayl mövcuddursa,daxil etdiyiniz yeni ifadələr,fayl daxilindəki əvvəlki ifadələri silmədən əlavə olunacaq.

w+ modu

Faylı oxumaq və yazmaq modunda açır.Sisteminizdə həmin adda fayl varsa daxilindəki ifadələr silinəcək,yoxdursa vrdiyiniz ada uyğun yeni fayl yaranacaqdır.

r+ modu

Bu modu istifadə etmək üçün sisteminizdə fayl mövcud olmalıdır.Faylı yazmaq və oxumaq modunda açır.

Yeni bir fayl açaraq ifadəmizi əlavə edək

```
>>> f=open('blachborn.txt','w')
>>> f.write('Python proqramlama dili')
>>> f.close()
>>> f
<_io.TextIOWrapper name='blachborn.txt' mode='w' encoding='UTF-8'>
```

Və fayl açıldığı qovluğa gedərək,həqiqətəndə ifadəmizin fayl daxilində

olduğuna əmin olduq.Əgər biz sonda f.close() ifadəsindən istifadə etməsəydik yazdığımız ifadə hələki fayla yazılmayacaqdı.
Ən əsası,faylı hansı mod'da açmağınıza diqqət edin.

Açdığımız fayla bir neçə ifadələr əlavə edək

```
>>> f=open('blachborn.txt','w')
>>> f.write('\nRubby proqramlama dili')
>>> f.write('\nPython proqramlama dili')
>>> f.close()
>>>
```

Açdığımız mətn faylından ifadələr oxuyaq

read() metodu

```
>>> f=open('blachborn.txt','r')
>>> f.read()
'\nRubby proqramlama dili\nPython proqramlama dili'
>>> f.read()
''
>>>
```

yuxarıda read() metodundan istifadə edərək fayldan ifadələri oxuduq.Amma ikinci dəfə biz read() metodundan istifadə edərək ifadələri oxuya bilmədik.Çünki ilk read() metodunu istifadə etdikdən sonra,sətrin sonuna düşdüyü üçün təkrar oxumaq mümkün deyil
Bunun üçün seek() metodundan istifadə edəcəyik.Metod bir argument alma imkanına malikdir.

```
>>> f.seek(0)
0
>>> f.read()
'Rubby proqramlama dili'
'Python proqramlama dili'
>>>
```

```
>>> f=open('blachborn.txt','w')
>>> f.write(i)
>>> f=open('blachborn.txt')
```

```
>>> print(f.read())
John  :  0553456789
Edgar :  0556789809
Malina :  0553452321
Blanca :  0513423231
>>> print(f.read())

>>> f.seek(0)
0
>>> print(f.read())
John  :  0553456789
Edgar :  0556789809
Malina :  0553452321
Blanca :  0513423231
>>>
```

fayllarda tell və seek yolu

Bir əvvəlki bəhsdə seek metodun nə rol oynadığını aç-çox başa düşdük.İndidə tell və seek metodlarından geniş bəhs edək
İlk əvvəl bir fayl yaradaq

```
>>> f=open('act.txt','a')
>>> f.write("""Şimaldan - Baş Qafqaz dağları, qərbdən - Göyçə gölü
hövzəsi də daxil olmaqla Alagöz dağ silsiləsi və Şərqi Anadolu, şərqdən -
Xəzər dənizi, cənubdan isə Sultaniyyə-Zəncan-Həmədan hüdudları ilə
əhatə olunan tarixi Azərbaycan torpaqları müasir sivilizasiyanın inkişafına
başladığı ən qədim mədəniyyət mərkəzlərindən biridir. Azərbaycan xalqı
bu ərazidə - tarixi Azərbaycan torpaqlarında zəngin və özünəməxsus bir
mədəniyyət, o cümlədən dövlətçilik ənənələri yaratmışdır.""")
468
```

Ilk əvvəl act.txt adlı fayl yaratdıq daha sonra bir mətni write metodu vasitəsilə fayla yerləşdirdik.

Və ardından ekranda 468 rəqəmi göründü.Bu əldə etdiyimiz rəqəm toplam mətnin bayt-həcmini ifadə edir.Bilirikki seek() metodundan istifadə edərək sətrin başına dönə bilərik.Qeyd edimki seek() metoduna verdiyimiz say-argumentləri əslində bayt-həcmini göstərir,yəni əgər seek(0) yazsaq 0-cı bayt,yəni sətrin başına qayıdacaq.
Biz tell() metodundan istifadə edərək hansı bayt-həcmin üstündə olduğunu öyrənə bilərik

```
>>> f.tell()
552
```

deməli faylda 552-bayt üzərindəyik

```
>>> f.seek(4)
4
```

yazaraq 4-baytlıq həcmə qayıtdıq,hər halda bu başlanğıc sətirlərdən biri olacaq

```
>>> f=open('act.txt')
>>> f.read()
'Şimaldan - Baş Qafqaz dağları, qərbdən - Göyçə gölü hövzəsi də daxil
olmaqla Alagöz dağ silsiləsi və Şərqi Anadolu, şərqdən - Xəzər dənizi,
cənubdan isə Sultaniyyə-Zəncan-Həmədan hüdudları ilə əhatə olunan
tarixi Azərbaycan torpaqları müasir sivilizasiyanın inkişafına başladığı ən
qədim mədəniyyət mərkəzlərindən biridir. Azərbaycan xalqı bu ərazidə -
tarixi Azərbaycan torpaqlarında zəngin və özünəməxsus bir mədəniyyət,
o cümlədən dövlətçilik ənənələri yaratmışdır.'
>>> f.seek(0)
0
>>> f.seek(5)
5
>>> f.tell()
5
>>>
```

r-modunda faylı açırıq,daha sonra 5-baytlıq sətrə seek(5) yazmaqla qayıdırıq.İndidə bunu əyani olaraq görək

```
>>> f.read()
'ldan - Baş Qafqaz dağları, qərbdən - Göyçə gölü hövzəsi də daxil olmaqla
Alagöz dağ silsiləsi və Şərqi Anadolu, şərqdən - Xəzər dənizi, cənubdan
isə Sultaniyyə-Zəncan-Həmədan hüdudları ilə əhatə olunan tarixi
Azərbaycan torpaqları müasir sivilizasiyanın inkişafına başladığı ən qədim
mədəniyyət mərkəzlərindən biridir. Azərbaycan xalqı bu ərazidə - tarixi
Azərbaycan torpaqlarında zəngin və özünəməxsus bir mədəniyyət, o
cümlədən dövlətçilik ənənələri yaratmışdır.'
>>>
```

Gördüyümüz kimi seek(5) verdiyimiz üçün 5-baytlıq həcmdən mətni oxuyaraq bizə çap etdi.'Şima hissəsi 4-baytlıq həcmdir

with metodu

with metodu vasitəsilə faylları bağlamadan əlavələrimizi edə bilərik.Metod əlavə olunan ifadələrdən sonra faylımızı bağlayır.Yəni bir növ close() metodunun əvəzləyicisidir.

Yeni bir fayl açaq

```
>>> with open('elips.lst','a') as f:
       f.write('texnoloq\nacademy\nwine')


21
>>>
```

Və fayldan ifadələri oxuyaq(bunun üçün faylı read(r) modunda açmalıyıq.

```
>>> with open('elips.lst') as f:
       print(f.read())


texnoloq
academy
```

```
wine
>>>
```

Və asanlıqla fayldan ifadələri oxuduq

Biz ifadələrimizi hər dəfə əlavə etdikdə bilirikki fayl daxilində son sətrə əlavə olunur.Əgər məcburi olaraq siz yeni əlavə etmək istədiyiniz ifadəni sətrin başında görünməsini istəsəniz,bunun üçün seek() metodundan istifadə edəcəksiniz

```
>>> with open('telbook.lst','a') as f:
	f.write('Ayşən  : 0556453421\nFuad : 0553423456')



37
>>> with open('telbook.lst','r+') as f:
	f.seek(0)
	f.write('Eldar : 0516678990')



0
18
>>>
```

Yuxarıdakı kodlarımızda isə biz son ifadəmizi əlavə etsəkdə Ayşən adlı ifadəmiz silindi.Bunun başqa yolu

```
>>> with open('telbook.lst','r+') as f:
	i=f.read()
	f.seek(0)
	f.write('Eldar  : 0556783421\n'+i)



0
57
>>>
```

Eldar : 0556783421
Ayşən : 0556453421
Fuad : 0553423456

Gördüyünüz kimi bu dəfə istəyimiz alındı.Kodlarımıza izahat gətirək

i=f.read() ifadəsini yazmağımızın səbəbi,növbəti nə dəyişiklik olursa olsun əvvəl yazdığımız ifadələr faylda qalmasını təmin edir.f.seek(0) -yazaraq sətrin əvvəlinə qayıdırıq ki əlavə edəcəyimiz ifadə ilk sətirdə olsun.f.write('Eldar : 0556783421\n'+i)-- ifadəsi isə əlavə etdiyimiz Eldar : 0556783421 -sonra i=f.read() əvvəlki faylda olan ifadələrimizi \n işarəsi vasitəsilə aşağı sətrə əlavə etsin.
\n -işarəsinin nə rol oynadığını biz öyrəndiyimiz üçün yəni Eldar : 0556783421 ifadəsi ilk sətrə əlavə olunsun daha sonra gələn ifadələr \n -xüsusi işarənin köməyi ilə aşağı sətirlərə əlavə olunsun.Aşağı sətirlərə də əlavələrimizi əvvəlki ifadələrimiz olan Ayşən və Fuad -ı əlavə etdik.

readline() metodu

```
>>> with open('telbook.lst') as f:
	print(f.readline())


Eldar  : 0556783421
```

```
>>> with open('telbook.lst') as f:
	f.readline()


'Eldar  : 0556783421\n'
>>>
```

readlines()   metodu

Biraz əvvəl biz read() metodundan istifadə edərək fayla ilk sətrə(0 baytlıq həcm) ifadə əlavə etdik.Bundan başqa digər sətirlərə də ifadə əlavə etmək istəsək əgər readlines() metodundan istifadə edəcəyik.readlines() metodu list metodu olduğu kimi fayllarda da sətirləri bir list daxilində oxuyur.bir iki üç və sairə,yəni hansısa sətrə ifadə əlavə etmək istəsək bu metoddan istifadə edə bilərik

```
>>> >>> with open('telbook.lst','r+') as f:
	i=f.readlines()
	i.insert(3,'eclipse\n')
	f.seek(0)
	for x in i:
		f.write(x)


0
>>>
```

Və fayla eclipse ifadəsini 3-cü sətrə əlavə etdik.
Biz ifadəmizi sonra for operatorundan istifadə edərək yazdırdıq.Bundan başqa birbaşa writelines() metodundan istifadə edərək yaza bilərik.
readlines() metodu bir list daxilində ifadəni oxuduğu kimi writelines() metoduda list daxilinə ifadə əlavə edir.

Və ya digər yolu ilk əvvəl readlines() metodundan istifadə edərək,fayl daxilində olan ifadələri oxuyub yadda saxlayırıq

```
>>> with open('telbook.lst') as f:
	i=f.readlines()
```

daha sonra faylımızı w- modunda açaraq

```
>>> with open('telbook.lst','w') as f:
	i.insert(1,'flamingo\n')
	f.writelines(i)
```

```
>>>
```

Növbəti ifadəmizi ikinci sətirə əlavə edirik.

Faylların digər metodları

Bu bəhsdə baxacağımız metodlar aşağıdakılardan ibarətdir

closed readable() writable() truncate()
mode name encoding

closed metodu

Metodun adından da bəlli olduğu kimi bağlı mənasını ifadə edir.
Metod faylın bağlı olub-olmadığını təyin edərək,ekrana iki argument çap edir True və False.Əgər fayl bağlıdırsa True,açıqdırsa False çap edəcəkdir.

```
>>> f.closed
True
>>>
```

Deməli faylımız hal-hazırda bağlıdır.
Əgər adı olmayan bir faylı bu metodla istifadə etsək xəta alarıq

```
>>> a.closed
Traceback (most recent call last):
  File "<pyshell#144>", line 1, in <module>
    a.closed
NameError: name 'a' is not defined
>>>
```

readable metodu

Dilimizə oxunaqlı kimi tərcümə olunur.Metod bir faylın r-modunda açılıb-açılmamasını təyin edir.

```
>>> f.readable
<built-in method readable of _io.TextIOWrapper object at
0x7f6ec3714630>
>>>
```

truncate() metodu

Dilimizə kəsmək,qırmaq kimi tərcümə olunur.Metod fayl daxilindəki ifadələri silir.

```
>>> with open('telbook.lst','w') as f:
        f.truncate()


0
>>>
```

Və telbook.lst adlı faylımıza baxdıqda içində olan məlumatların hamısı silinmişdir.

Metod say argumentləri ilə də istifadə olunur.Bu say-argumenti bayt-həcmlərini ifadə edir.Əgər biz f.truncate(3) yazsaydıq metod 3-baytlıq ifadəyə toxunmayacaq,digər ifadələri siləcəkdir.

mode metodu

Metod bizə faylın hansı modda açıldığı haqqında məlumat verərək,birbaşa ekrana modu çap edir

```
>>> f.mode
'w'
>>>
```

name metodu

Metodun tərcüməsindən də aydın olurki bir şeyin adını ifadə etmə xüsusiyyətinə malikdir.Beləki metod faylın adını ekrana çap edirik.python3

```
>>> f.name
'telbook.lst'
>>>
```

encoding metodu

Dil kodlaması ilə bağlı metod növüdür.

```
>>> f.encoding
'UTF-8'
>>>

>>> f.encoding
'cp1254'
```

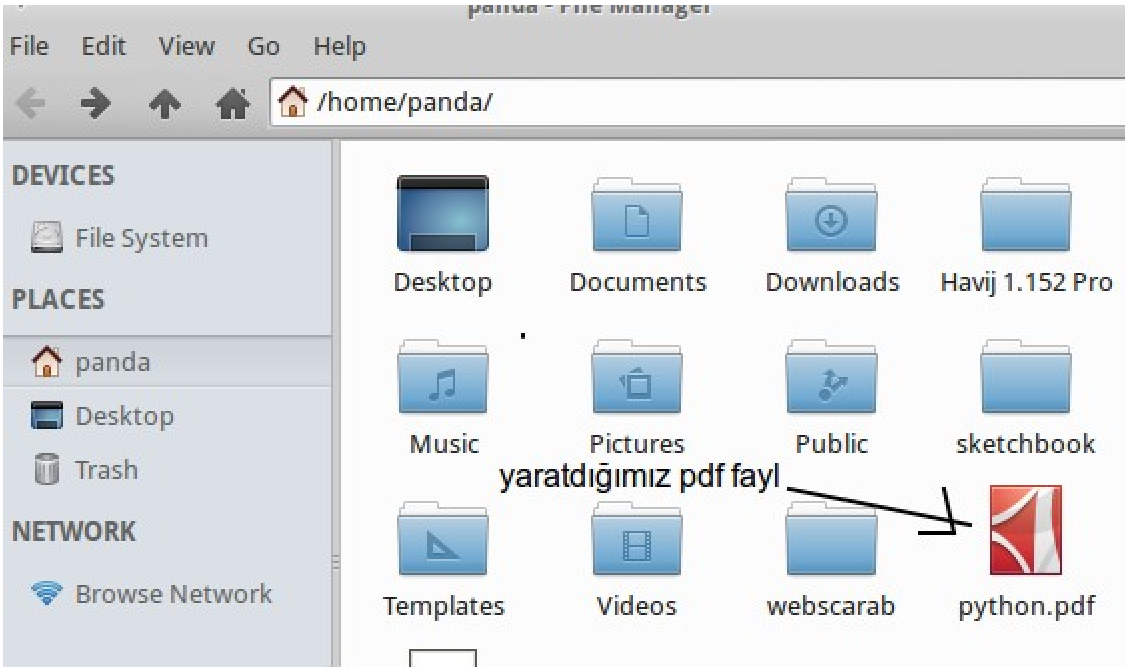
Binar fayllar

bura qədər kodlarımızda istifadə etdiyimiz fayllara(sonu txt,lst və s),rahatlıqla giriş edib daxilindəki ifadələri oxuya bilirdik.Bunlardan başqa bəzi fayl tipləri mövcuddurki,faylı açarkən qarışıq simvollarla rastlaşa bilərik.Bu tip fayllar binar fayllar adlanır.Bunlara misal olaraq,bu gün gündəlik istifadə etdiyimiz,mp3,mp4,GİF,jpg,png,doc,pdf ,dat,bin kimi faylları misal göstərmək olar.Bu faylların istifadə etdiyi fayl modları da tamam fərqlidir.

Misal üçün bir pdf faylı yaradaq,daha sonra bu fayldan ifadələr oxumağa çalışaq.

```
>>> f=open('python.pdf','ab+')
```

və aşağı sətrə xətasız keçdik.Daha sonra python-u çalışdırdığımız qovluğa gedib(adətən bu Home-user qovluğunda olur.)açıldığına əmin olaq.



İndidə yuxarıdakı kodlarımıza açıqlıq verək

`f=open('python.pdf','ab+')` kodlarımız daxilində ab+ fayl modudur.

a-modu bizə məlumdur.b isə b- binary,binar fayllarda istifadə olunur.

ab+ modu birlikdə binar faylı həm oxuma,həm də yazma modunda açır.

Daha sonra açdığımız pdf-formatındakı faylımızdan ifadələr oxuyaq

```
>>>f.read(1)
b''
>>>
```

Əvəlki bəhslərdə ilk başda qeyd etdiyimiz modlara sadəcə b(binary)əlavə edib,binar fayllar üçün istifadə edəcəyik.Öncəki modların funksional işi binar fayllarda da dəyişməz olaraq qalacaq.
binar fayllar üçün istifadə olunan modları toplu şəkildə nəzərdən keçirək

rb modu

r modunda olan (read-oxumaq) funksional fəaliyyəti binar-fayllar üçün də istifadə olunur.Yəni biz bir faylı f=open('yenifayl.txt','r') kimi açırdıqsa,binar-fayllarımızı da rb-modunda açıb,daxilindən ifadələri oxumaq üçün istifadə edəcəyik.

rb+ modu

binar-faylını həm oxuma,həm də yazma modunda açır.

wb modu

binar-faylını yazma modunda açır

wb+ modu

binar-faylını həm oxuma,həm də yazma modunda açır

ab modu

binar-faylını yazma modunda açır

ab+ modu

binar-faylını həm oxuma,həm də yazma modunda açır

xb modu

binar-faylını yazma modunda açır

xb+ modu

binar-faylını həm oxuma,həm də yazma modunda açır

```
>>> f=open('functional-python-programming.pdf','rb')
>>> f.read(1)
b'%'
>>> f.read(40)
b'PDF-1.6\r%\xe2\xe3\xcf\xd3\r\n17686 0 obj\r<</Linearized'
>>> f.read(60)
b' 1/L 2324392/O 17688/E 146096/N 361/T 2321140/H [ 476 5464]>'
```

Bunun üçün gəlin internetdən bir kitab yükləyərək kitab haqqında bəzi məlumatları alaq.Qeyd edimki bir elektron kitabın sabit ifadələri varki,bu ifadələr vasitəsilə kitabın yazıldığı tarix,müəllif və kitaba sponsorluq edən şirkət haqqında məlumat ala bilərik.Bu hər e-kitabda özünü biruzə verə bilməz.Çünki bir pdf formatında kitab yazmağa çalışsanız,kitabı pythonla həmin sabit ifadələrdən istifadə edərək bəzi məlumatları daxil edə bilərsiniz.

Kitabı yükləmək üçün linux-un bizə təklif etdiyi wget komandasından(əmrindən) istifadə edərək

```
$ pwd
/home/user
$ wget
http://ptgmedia.pearsoncmg.com/images/9780132678209/samplepages/
0132678209.pdf
--2016-10-22 17:04:31--
http://ptgmedia.pearsoncmg.com/images/9780132678209/samplepages/
0132678209.pdf
Resolving ptgmedia.pearsoncmg.com (ptgmedia.pearsoncmg.com)...
23.50.183.9
Connecting to ptgmedia.pearsoncmg.com (ptgmedia.pearsoncmg.com)|
23.50.183.9|:80... connected.
```

```
HTTP request sent, awaiting response... 200 OK
Length: 11213483 (11M) [application/pdf]
Saving to: ‘0132678209.pdf’

100%[=====================================>]
11.213.483   172KB/s   in 2m 3s

2016-10-22 17:06:35 (88,7 KB/s) - ‘0132678209.pdf’ saved
[11213483/11213483]
$
```

və python shell açırıq(idle3 və sairə)

kitabın adını dəyişib core_python.pdf olaraq qeyd etdim.

```
>>> f=open('core_python.pdf','rb')
```

yazaraq yüklədiyimiz kitaba oxuma modu veririk

```
>>> i=f.read()
>>> producer=i.index(b'/Producer')
>>> producer
296697
>>> i[producer]
47
```

Yuxarıda index metodundan istifadə edərək Producer-(odt,word və altındakı yazıları pdf -ə çevirən sistem haqqında məlumat verir) sabit ifadəsinin bayt tutumunu əldə etdik. sıra nömrəsi isə demək olarki bütün pdf kitablarda eyni nəticəni verir.
b'/Producer' -əvvəlinə yazdığımız b-hərfi isə bayt mənasını daşıyır,yəni Producer ifadəsinin b-baytını əldə etmək üçün istifadə etdik.
Indidə kitabdan baytı yazaraq daxilindəki ifadəni oxumağa çalışaq

```
>>>i[producer:producer+60]
b'/Producer(PDFKit.NET 2.0.28.0)/Subject()/Title(Core Python A'
```

və qarşımıza yuxarıdakı ifadələri çıxardı

İfadələrimizi split metodundan istifadə edərək bir list halına salaq

```
>>> i[producer:producer+60].split()
[b'/Producer(PDFKit.NET', b'2.0.28.0)/Subject()/Title(Core', b'Python', b'A']
```

Daha sonra yuxarıdakı kodları digər dəyişənə qeyd edərək

```
>>> producer=i[producer:producer+60].split()

>>> producer
[b'/Producer(PDFKit.NET', b'2.0.28.0)/Subject()/Title(Core', b'Python', b'A']
>>>

>>> producer[0]
b'/Producer(PDFKit.NET'
>>> producer[1]
b'2.0.28.0)/Subject()/Title(Core'
>>> producer[2]
b'Python'
>>> producer[3]
b'A'
```

b'/Producer(PDFKit.NET'   -0
b'2.0.28.0)/Subject()/Title(Core'   -1
b'Python'   -2
b'A' -3

```
>>> f=open('core_python.pdf','rb')
>>> i=f.read()
>>> title=i.index(b'/Title')
>>> title
296759
>>> i[title]
47
>>> i[title:title+100]
b'/Title(Core Python Applications Programming)/Universal#20PDF(The
process that creates this PDF const'
>>>
```

title-başlığımız

Core Python Applications Programming

Yuxarıda istifadə etdiyimiz sabit ifadələrə toplu baxaq

/Creator -sistem haqqında məlumat verir(Nt,MacOs və ya Posix)

/Producer -hansı proqramda pdf-formatına çevrilib

/Title -kitabın başlığı

/Author -kitabın müəllifi

/Subject -kitabın mövzusu

/Keywords -açar sözlər

/CreationDate -kitabın yazıldığı tarix

/ModDate   -dəyişdirilmiş tarixi

ASCII kodlaması

Baş hərfləri uyğun olaraq

A-American
S-Standart
C-Code
İ-İnformation

İ-İnterchange

sözlərin dilimizə tərcüməsi

məlumatın mübadiləsi Amerika standart kodları ilə
Bu dil qrupu Bob Bemer tərəfindən yaranıb.Kodlamada əsasən saylara uyğun ifadələr qeyd olunub.Son rəqəmi 128-də bitir.Standart olaraq 7-bitlik bir sistemdir.Cədvəl http://www.asciitable.com/ ünvanında qeyd olunub.Əgər nəzər yetirsəniz.Bu 7-bitlik kodlama sistemində yalnız İngilis əlifbası və dəyişilməz riyazi,şərti işarələr qeyd olunub.Yəni 32-dən 128-ə qədər olan saylara istifadə edəcəyimiz hərflər və işarələr qeyd olunub.Python2-də tez-tez rastlaşdığımız dil problemlərindən biridə Azərbaycan əlifbasında olan ə,ğ,ö,ü,ı,ş və s hərflərin qarşılıq olaraq fərqli ifadələr çap olunurdu.Və ya xəta ilə qarşılaşaraq

```
File "skype.py", line 1
SyntaxError: Non-ASCII character '\xfe' in file skype.py on line 1, but no encoding declared; see http://www.python.org/peps/pep-0263.html for details
```

xətada deyildiyi kimi yazdığımız ifadə ascii-kodlamasından kənardadır.
Bu kodlama sistemi bu gün geniş istifadə olunmaqdadırki parolların istifadəçidən alınması,url-səhifə başlıqları,məktublaşmalarda dialogun standart məlumat mübadilələri və s misal çəkmək olar.
Dilimizdə olan yuxarıda saydığım hərflərə qarşılığı pythonda təyin edə bilərik.Məsələn

```
>>> 'ə'.encode('utf-8')
b'\xc9\x99'
```

utf-8 sistemində qarşılıqlı ifadəsi '\xc9\x99' uyğun gəlir.Amma biz ə – hərfini ascii-7 bitlik sistemdə sorğuya çəksək

```
'ə'.encode('ascii')
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    'ə'.encode('ascii')
UnicodeEncodeError: 'ascii' codec can't encode character '\u0259' in position 0: ordinal not in range(128)
```

xəta ilə qarşılaşacağıq.Digər dillərin əlifbasından istifadəsi üçün daha alternativ yol fikirləşərək bu kodlamanı 128-256 aralığına qaldıraraq əlavələrini etdilər.

Unicode

özündən əvvəlki ascii -kodlamasını bazasına daxil etməklə əlavələr edərək daha böyük bit həcminə malikdir.Bu sistem bir milyondan artıq ifadələrin qarşılığını bazasında daşıyaraq kodlayır.İlk dəfə 16-bit olaraq yaradılmışdır.Bunun riyazi olaraq ifadə etsək 2**16=65536 sayına qarşılıq gəldiyindən bazasında nəqədər kodlamanın olduğunu təsəvvür etmək və ya onları burda bir cədvəldə qeyd etmək uzun iş olardı.Unicode öz növbəsində ascii -dən fərqli olaraq argumentin qarşılıq sayını birbaşa çap etmir.

0063 'c' qarşılıq gəldiyindən

```
>>> int('63',16)
99
>>>
```

onaltılıq say sistemində qarşılığı 99-a uyğun gələcəkdir

Bu kodlamalardan başqa UTF-1, UTF-7, UTF-8, UTF-16 ve UTF-32 bitlik kodlama sistemləri də var.
Bu sistemlərin hər birinə aid qarşılıqlı cədvəlləri mövcuddurki bunları internetdən asanlıqla əldə edə bilərik.

Kodlamalarda istifadə olunan parametrlər

strict
ignore
replace
xmlcharrefreplace
backslashreplace
namereplace

```
>>> 'Azərbaycan dövləti'.encode('utf-8',errors='strict')
b'Az\xc9\x99rbaycan d\xc3\xb6vl\xc9\x99ti'
```

istifadə etdiyimiz dilimizdəki hərflərin utf-8,8-bitlik kodlama sistemində qarşılığını çap etdi

və ya

```
>>> 'Azərbaycan dövləti'.encode('utf-16',errors='strict')
b'\xff\xfeA\x00z\x00Y\x02r\x00b\x00a\x00y\x00c\x00a\x00n\x00
\x00d\x00\xf6\x00v\x00l\x00Y\x02t\x00i\x00'
>>>
```

və ya

```
>>> 'Azərbaycan dövləti'.encode('utf-16','strict')
b'\xff\xfeA\x00z\x00Y\x02r\x00b\x00a\x00y\x00c\x00a\x00n\x00
\x00d\x00\xf6\x00v\x00l\x00Y\x02t\x00i\x00'
>>>
```

əgər 7-bitlik ascii kodlamasını sorğuya çəksək

```
>>> 'Azərbaycan dövləti'.encode('ascii',errors='strict')
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    'Azərbaycan dövləti'.encode('ascii',errors='strict')
UnicodeEncodeError: 'ascii' codec can't encode character '\u0259' in
position 2: ordinal not in range(128)
```

xəta ilə qarşılaşacağıq.Çünki istifadə etdiyimiz hərflər(dilimizdə olan) 7-bitlik sistemdə yoxdur.

Əgər xətanı ignore etsək

```
>>> 'Azərbaycan dövləti'.encode('ascii',errors='ignore')
b'Azrbaycan dvlti'
>>>
```

ascii-cədvəlində olmayan yazdığımız hərfləri çapda göstərməyərək xəta verməyəcək

replace parametrini istifadə edərək,kodlama sistemində olmayan ifadələri ? -işarəsi ilə əvəz edə bilərik

```
>>> 'Azərbaycan dövləti'.encode('ascii',errors='replace')
b'Az?rbaycan d?vl?ti'
>>>
```

encoding parametri

Əvvəlki bəhslərdə faylları öncədən modunu qeyd edib açırdıqsa,bu parametri əlavə edərək, faylı hansı dil kodlaması ilə açacağımızı qeyd edə bilərik.Bunun üçün

```
>>> f=open('cprogramming.pdf',encoding='utf-8')
>>> f.read(20)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    f.read(20)
  File "/usr/lib/python3.4/codecs.py", line 319, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb5 in position 11: invalid start byte
>>>
```

yazaraq xəta aldıq,yəni bu kitab başqa kodlama sistemi ilə açıldığından utf-8 ilə xəta verir.
Errors funksiyasının aldığı parametri burda da istifadə edə bilərik

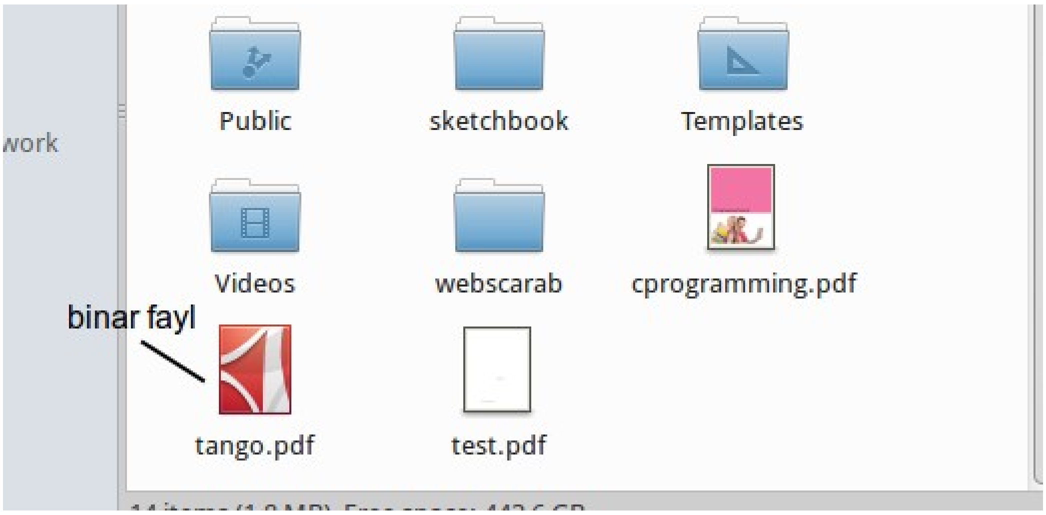
```
f=open('cprogramming.pdf',encoding='ascii',errors='strict')
f=open('cprogramming.pdf',encoding='ascii',errors='replace'
f=open('cprogramming.pdf',encoding='ascii',errors='ignore')
```

yalnız ‘xmlcharrefreplace’ parametri open funksiyası daxilində istifadə edilə bilməz.

Yeni bir binar fayl açaraq

```
>>> f=open('tango.pdf','ab')
>>> f=open('tango.pdf',encoding='utf-8')
```

dil kodlamasını utf-8 olaraq qeyd etdik



Kodlama sistemlərində istifadə olunacaq funksiyalar

repr()

repr funksiyası ifadəni olduğu kimi çap edir.Yəni bir ifadənin hansısa kodlama sisteminə qarşılığını çap etmir.Misal üçün

```
>>> repr('i')
"'i'"
>>> repr(3)
'3'
>>>
```

Daha açıq desək ifadəni olduğu kimi string cinsinə çevirərək çap edir.
ascii

Funksiya kimi ascii bir ifadənin unicode qarşılığını göstərir.repr() funksiyasından fərqi,repr() ifadəni olduğu kimi çap edirdisə,ascii() isə onun kodlama sistemindəki qarşılığını çap edəcək.Misallara baxaq

```
>>> ascii('ə')
"'\\u0259'"
>>> ascii('ğ')
"'\\u011f'"
>>>
```

ord() funksiyası

Bu funksiya ifadənin ascii cədvəlindəki chr qarşılıq gəldiyi sayını göstərir

```
>>> ord('a')
97
>>> ord('\n')
10
```

cədvələ bu http://www.asciitable.com/ ünvandan baxa bilərsiniz

chr() funksiyası

Necə ord() funksiyası vasitəsilə ifadənin cədvəldəki sayını təyin edirdiksə,bu funksiya vasitəsilə saya uyğun gələn ifadəni təyin edə bilərik

```
>>> ord('\n')
10
>>> chr(10)
'\n'
>>>
```

bayt(bytes)

binary digit sözündən götürülərək dilimizə ikili say(saylar) kimi tərcümə olunur.bayt 8 -bitdən şərti olaraq qeyd edilir.Bura qədər keçdiyimiz faylların hər biri müəyyən bit və ya bayt həcminə malikdir.Yəni istifadə etdiyimiz hesablama maşınları,kompyuterlər,robot qurğuları bütünlükdə daxilində olan mikroidarəedicilərin içində rəqəmlərlə təşkil olunaraq bizə öz bacarıq funksiyalarını göstərir.Təbidirki bu əmrlər bütünlükdə hər hansı bir proqramlama dilində yazılaraq icraediciyə yüklənir.Bütün bunların başında insan əməyi,zəkası dayanır.mahiyyət etibarı ilə insan dilinə yaxın olan kod sistemləri,maşınlarda sadəcə saylardan təşkil olunub.
Biz əvvəlki bəhslərimizdə ascii kodlama sistemindən danışmışdıq.Və mənbələrə əsasən qeyd etdikki bu 128-ə qədər saylardan təşkil olunaraq ingilis dili əlifbasını eləcədə işarələri bir cədvələ toplayaraq bir dil kodlama sistemi yaratmışdır.ascii kodlama sistemi 7-bit olduğunu qeyd

etmişdik.

```
>>> 'ə'.encode('utf-8')
b'\xc9\x99'
>>>
```

Yuxarıda ə -hərfini utf-8 sistemində qarşılığını sorğuya çəkdik.utf-8,8 bitlik və 1 baytlıq sistemdir.Yuxarıdakı nəticədən bizə lazım olacaq c999 ifadəsidirki 2baytlıq (16-bit) qarşılığını tapaq.

```
>>> int('c999',16)
51609
>>>
```

və python bizə 51609 cavabını verdi.Geriyə dönüb neçə bitdən təşkil olunduğunu sorğuya çəksək bit_length() metodundan istifadə edəcəyik

```
>>> (51609).bit_length()
16
>>>
```

16 bit,2 baytdan təşkil olunub.

Boş bir bayt

```
>>> bayt=b''
>>> type(bayt)
<class 'bytes'>
>>>
```

Python shell-i açırıq

```
>>> dir(bytes)
```

yazaraq

və ya

```
>>> for i in dir(bytes):
        if '_' not in i:
                print (i)
```

baytların istifadə edə biləcəyi metodları çap edirik.Gördüyümüz kimi str cinsində olan ifadələrin istifadə edə biləcəyi metodları,bayt ilə də istifadə edə bilərik.Daha ətraflı

```
>>> help(bytes)
```

yazaraq,istifadə qaydası ilə tanış ola bilərsiniz.Deməli string cinsində olan ifadələr həmdə bit və baytdır.

Python shelld-dən istifadə etdiyimiz dil kodlamasını sorğuya çəkək

```
>>> import locale
>>> locale.getpreferredencoding()
'UTF-8'
```

Və hal-hazırda istifadə etdiyimiz utf-8 kodlama sistemi üzərindəyik

Bu kodlama sistemində olan str cinsərini sorğuya çəksək

```
>>> b'a'
b'a'
```

olduğu kimi ekranda çap olunacaqdır.

```
>>> b'ə'
SyntaxError: bytes can only contain ASCII literal characters.
```

Son nəticədə xəta aldıq.

Ifadənin əvvəlinə böyük B hərfi də yazsanız nəticəni eyni əldə edəcəksiniz

```
>>> B'a'
b'a'
>>>
```

Unikod kodlama cədvəlinə daxil olan ifadələri say sistemlərinə görə yazaraq ifadəni əldə edə bilərik.baytlar 0-256 aralığında 0 və 255 daxil olmaqla saylara uyğun kodlama sistemidir

```
>>> bytes([96,97,98,99])
b'`abc'
>>>
```

Gördüyünüz kimi saylara daxil olan ifadələri çap etdirdik.

Əgər Unikod kodlama sisteminə daxil olmayan sayları qeyd etsəniz xəta alacaqsınız
İlk əvvəl qeyd etdiyimiz kimi str cinslərinin istifadə etdiyi metodları bytes() funksiyasında da istifadə edə bilərik

```
>>> i=b'Azerbaijan'
>>> i
b'Azerbaijan'
>>> i[1]
122
>>> i[1:4]
b'zer'
>>> len(i)
10
>>> type(i)
<class 'bytes'>
>>>

>>> url=b'www.'
>>> url1=b'python.org'
>>> url+url1
b'www.python.org'
>>>
```

decode() metodu

encode() metodunu keçərkən əldə etdiyimiz kodlama sistemindəki qarşılığını decode() metodu əksinə olaraq qarşılıqlı ifadəni göstərir.

```
>>> 'Ö'.encode('utf-8')
b'\xc3\x96'
>>> 'Ö'.encode('utf-16')
```

```
b'\xff\xfe\xd6\x00'
>>> b'\xc3\x96'.decode('utf-8')
'Ö'
>>> b'\xff\xfe\xd6\x00'.decode('utf-16')
'Ö'
>>>
```

fromhex() metodu

Metod onaltılıq say sistemindəki ifadəni bayt-a çevirir

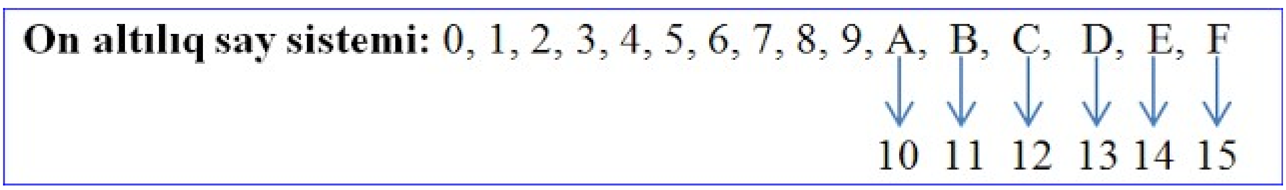
```
>>> bytes.fromhex('6c 6d ')
b'lm'
>>>
```

ötürülən boşluqlar nəzərə alınmır.

Say sistemləri

Say sistemlərinə texnologiyada baza elementləri də deyirlər.Say sistemləri iki yerə bölünür
1 Mövqeli
2 Mövqesiz

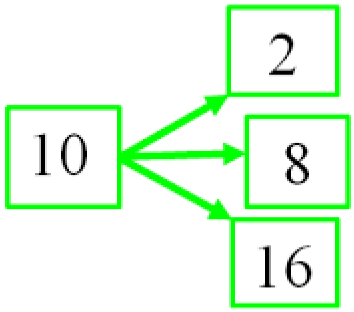
Bundan başqa elementlərin təsnifatına görə ikili,onlu,onaltılıq say sistemləri də vardır.
Adlarından da göründüyü kimi hər bir say sistemi bazasında olan elementlər qədərdir.Məsələn ikili say sistemləri yalnız 0 və 1 rəqəmlərindən təşkil olunub.
Səkkizlik say sistemləri isə 0,1,2,....8
rəqəmlərindən təşkil olunub.
Onluq say sistemi: rəqəmləri: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 on rəqəmi olduğuna görə onluq say sistemi adlanır.
Onaltılıq say sistemləri isə



On altılıq say sistemində 16 rəqəm var. 0-dan 9-a qədər olan rəqəmlər onluq say sistemində olduğu kimidir. Digər rəqəmlər isə hərflərlə işarələnib.
rəqəm ilə ədəd anlayışını qarışdırmayın.ədədləri rəqəmlər,simvollar təşkil edir.Məsələn 15 ədədi iki simvoldan(rəqəmdən) 1 və 5-dən təşkil olunub.

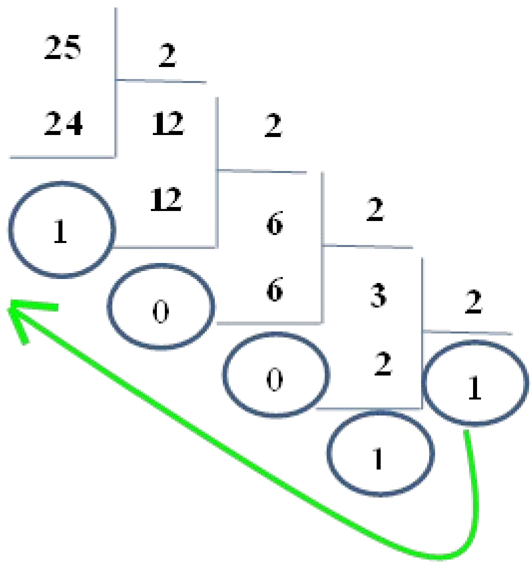
Say sistemlərindən digərinə keçid

onluq say sistemindən ikilik (səkkizlik, onaltılıq) say sisteminə keçmək üçün ədədi ikiyə (səkkizə, on altıya) bölmək və qalığı qeyd etmək lazımdır. Sonra aldığımız cavabı yenidən ikiyə (səkkizə, on altıya) bölüb qalığı qeyd edirik. unun axıra kimi davam etdiririk. Sağdan sola ardıcıl qalıqları yazırıq



Nümunə üçün 25 ədədini ikili sy sisteminə çevirək.Bilirikki 25-ədədi onluq say sisteminə aiddir

Şəkildən göründüyü kimi ilk əvvəl 25-ədədini 2-ə bölürük.Cavab 12, qalıq isə 1-dir.Eyni qayda ilə 12-cavabını 2-yə bölərək son nəticə alana qədər bu prosesi davam etdiririk.

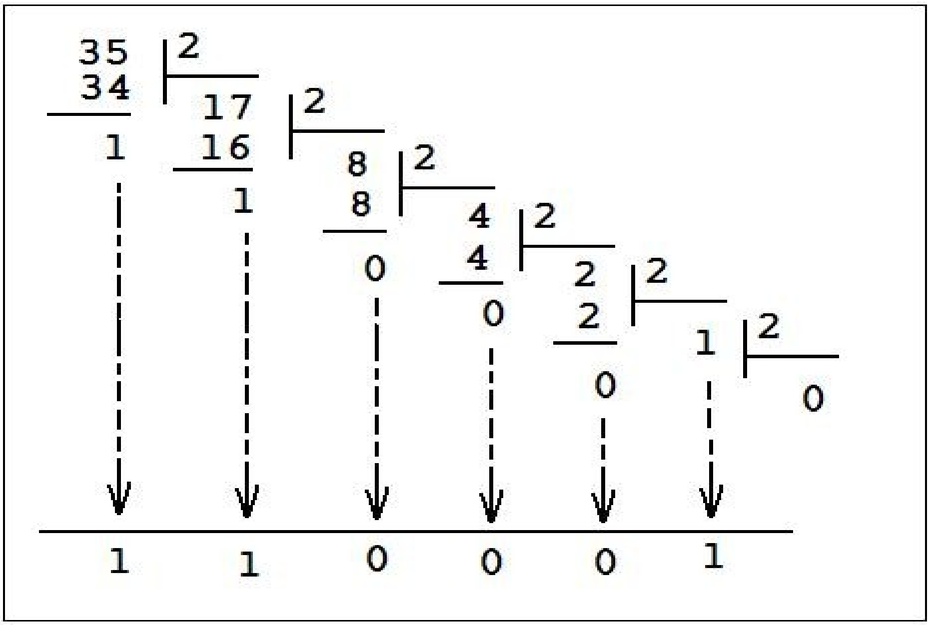


Və əldə etiyimiz 10011 qalıqları bizə lazım olan cavabdır.Deməli 25 ədədi

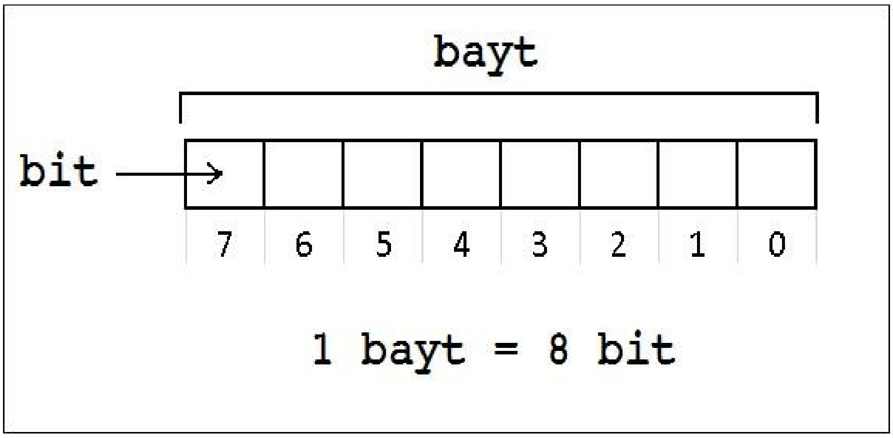
ikili say sistemində görünüşü 10011 -kimidir.
Yenidən bu ikili say sistemini geri dönərək onluq say sisteminə çevirmək üçün
baza elementlərini hecalara bölmək lazımdır.

110012 ədədininin sağdan başlayaraq üzərinə 0,1, 2, 3, 4 rəqəmlərini yazırıq. Sonra isə rəqəmləri 2-nin uyğun qüvvətinə vurub toplayırıq

onluq say sisteminə daxil olan 35 ədədinin ikili say sistemdə görünüşü



əldə etdiyimiz cavab 100011-dir



Şəkildən gördüyünüz kimi səkkizlik say sistemində bir bayt səkkiz bitlik yer tutur

Bunlardan başqa python bizə say sistemlərini digərlərinə çevirmək üçün funksiyalar təklif edir.

bin() funksiyası

Funksiya sayın,ikili say sistemində qarşılığını göstərir

```
>>> bin(2)
'0b10'
>>> bin(13)
'0b1101'
>>> bin(25)
'0b11001'
>>> bin(13)[2]
'1'
>>> bin(13)[1]
'b'
>>>
```

hex() funksiyası

Funksiya bir sayı onaltılıq say sistemində qarşılığını verir

```
>>> hex(4)
'0x4'
>>> hex(20)
'0x14'
>>> hex(20)[1]
'x'
```

```
>>>
```

oct() funksiyası

oktedlik yəni səkkizlik mənasını ifadə edir.Sayını səkkizlik say sistemində qarşılığını göstərir

```
>>> oct(2)
'0o2'
>>> oct(45)
'0o55'
>>> oct(45)[1]
'o'
>>> oct(45)[2]
'5'
>>>
```

int() funksiyası

int tam ədədlərlə çalışdığı məlum olsada bundan başqa sayının onluq sistemdə qarşılığını göstərir.İki argument alır

```
>>> int('4bc',16)
1212
>>> int('3df',16)
991
>>>
```

format() metodu ilə
xüsusi hərflər

b -binary ikili say sistemləri üçün
x -hexal onaltılıq say sistemləri üçün
o -oct səkkizlik say sistemləri üçün

```
>>> '{:b}'.format(24)
'11000'
```

```
>>> '{:x}'.format(2002)
'7d2'
>>> '{:o}'.format(2002)
'3722'
>>>
```

Pythonda təməl funksiyalara toplu baxış (built -in function)

Dilimizə tərcüməsi qurulu,hazır,özül,təməl kimi tərcümə edə bilərik. özül funksiyalar aşağıdakışəkildə göstərilmişdir.

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

özül funksiyalara https://docs.python.org/3/library/functions.html ünvandan ətraflı baxa bilərsiniz

abs() funksiyası

İngiliscədə absolute sözünün qısaltmasıdır.Dilimizə mütləq,əminlik mənasında tərcümə olunur.Funksiya həm tam ədədlər,həm də kəsirli

ədədlərin-ümumilikdə ədədlərin mütləq qiymətini verir.Riyaziyatda mütləq qiymət-həqiqi ədədin işarəsiz qiymətidir.

```
>>> abs(20)
20
>>> abs(-20)
20
>>> abs(20.4)
20.4
>>> abs(-20.4)
20.4
>>>
```

Qarışıq ədədlər-complex ədədlərə də bunu aid etmək olar

```
>>> complex(3)
(3+0j)
>>> abs(3+0j)
3.0
>>> complex(2.0)
(2+0j)
>>> abs(2+0j)
2.0
>>>
```

İlk əvvəl ədədi complex cinsə çevirdik daha sonra qarışıq ədədin mütləq qiymətini abs() funksiyası vasitəsilə soruşduq.

round() funksiyası

round dilimizə ətraf,dəyirmi,yuvarlaq kimi tərcümə olunur.Funksiya pythonda bir ədədi yuvarlaqlaşdırır.

```
>>> round(12)
12
>>> round(12.5)
12
>>> round(12.6)
```

```
13
>>> round(12.9)
13
>>>
```

Riyaziyyatdan da bizə məlumdurki kəsirli ədədlərdə nöqtədən sonra əgər .5 gələrsə ədəd yuvarlaqlaşdırırlmır,əgər .6,.7,.8 və s gələrsə ədəd yuvarlaqlaşdırılır.

round() funksiyası iki argument ala bilir.Bu argumentlər əsasən funksiyanın daxilində hesablama əməliyyatıdır.Yəni ilk öncə funksiya daxilindəki əməli yerinə yetirərək sonda əldə olunan cavabı-öz icra etdiyi funksional sistemlə yerinə yetirir.Misallara baxaq

```
>>> 34/2
17.0
>>> round(17.0)
17
>>> round(34/2)
17
>>> 14//9
1
>>> round(14//9)
1
>>> round(14/9)
2
>>>
```

Ikinci ala biləcəyi argument isə nöqtədən neçə sayı yuvarlaqlaşdırma əmridir.

```
>>> round(14/9,2)
1.56
>>> round(23//34,3)
0
>>>
```

all() funksiyası

Funksiya dilimizə bütün kimi tərcümə olunur.bool funksiyasına bənzəyir,yəni bir verilənin doğru olub-olmadığını test edir.Verdiyi cavabları True(təsdiq) və False(inkar)

```
>>> bytes=b''
>>> all(bytes)
True
>>> lst=[]
>>> all(lst)
True
>>> lst=[1,2,""]
>>> all(lst)
False
>>> dict={}
>>> all(dict)
True
>>>
```

ala biləcəyi xəta

```
>>> a=1
>>> b=2
>>> c=a>b
>>> all(c)
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    all(c)
TypeError: 'bool' object is not iterable
>>>
```

bunun üçün qapalı mötərizədən istifadə edərək

```
>>> all([c])
False
>>> a=1
>>> b=2
>>> c=b>a
>>> all([c])
True
```

```
>>>
```

any() funksiyası

Dilimizə 'hər hansı biri' kimi tərcümə olunur.Funksiyanın pythonda rolu digər funksiyaların daxilindəki ifadələrdən ən az biri düzgün yazılmalıdır. Əgər bir neçə ifadədən sadəcə biri doğrudursa True,tamamilə yanlışdırsa False verəcək.
Funksiya həm də qarşılaşdırma funksiyasını daşıyır

```
>>> dict={'alma','edam','',3}
>>> any(dict)
True
>>>
```

```
>>> list = [2,-3,-4,5,6]
>>> i= any(x>0 for x in lst)
>>> print (i)
True
>>>
```

Yuxarıdakı kodlarımızda list daxilində qeyd etdiyimiz ifadələrdən 2,5,6 ədədləri 0-dan böyük,-3,-4 ədədləri isə 0-dan kiçikdir.
any() funksiyası sadəcə ifadələrdən ən az birinin bool dəyərinin doğru olduğuna baxaraq True verir,əgər bütünlükdə bool dəyərləri yanlış olarsa False verəcək.Bu şərtlərə uyğun olaraq 2,5,6 ədədləri 0-dan böyük olduğu üçün True verdi.Yəni funksiya ilk öncə True dəyərinə baxıb sonra False dəyərini yoxlayır.
İndidə False verəcək ifadələr yazaq

```
>>> list = [2,3,4,5,6,7]
>>> i=any(x<0 for x in lst)
>>> print(i)
False
>>>
```

Və gördüyümüz kimi list-daxilində 0-dan kiçik olan ədəd yoxdur,ona görədə False çap etdi.

ascii() funksiyası

Bu funksiyadan kodlama sistemində istifadə etmişdik.Misallara baxaq

```
>>> ascii('\n')
"'\\n'"
>>> ascii('texnologiya')
"'texnologiya'"
>>> ascii('Qarabağ')
"'Qaraba\\u011f'"
>>>
```

Son qarabağ verilənində ğ-hərfi ascii-dil kodlama sistemindən kənarda olduğu üçün ekrana u011f-ifadəsini çap etdi.

bin() funksiyası

Funksiya verilənin ikili say sistemində qarşılığını verir.binary-ikili mənasını ifadə edir.

```
>>> bin(12)
'0b1100'
>>> bin(23)
'0b10111'
>>>
```

Funksiya yalnız tam ədədlər alır

```
>>> bin(13.0)
Traceback (most recent call last):
  File "<pyshell#73>", line 1, in <module>
    bin(13.0)
TypeError: 'float' object cannot be interpreted as an integer
```

```
>>> bin('Qarabağ')
Traceback (most recent call last):
  File "<pyshell#71>", line 1, in <module>
    bin('Qarabağ')
TypeError: 'str' object cannot be interpreted as an integer
```

bool() funksiyası

İfadə edə biləcəyi iki True və False var.Boşluq və 0-dan başqa istənilən verilənin dəyəri True-təsdiqdir

```
>>> bool(a)
True
>>> bool(1)
True
>>> bool( ) #   -boşluq
False
>>> bool('') # ' ' -boşluq
False
>>> bool(0) #0-ədədi
False
>>> bool('qutan')
True
>>> bool('aksioma')
True
```

bytearray() funksiyası

Bu funksiya həmdə bayt massivi adlanır.Baytlar dəyişilməyən tip olduğundan python bizə bayt massivindən(bytearray())istifadə etməyi təklif edir.baytlarda bir ifadənin daxilindəki verilən dil kodlamasında hansı rəqəmə uyğun gəlirsə sonradan başqa bir verilənlə dəyişdirmək olmur.Yəni azərbaycan sözünü baytla qeyd etsək,daha sonra bu kiçik a-nı böyük A-ilə dəyişməyə çalışsaq xəta alacağıq

```
>>> bytes=b'azerbaijan','ascii'
>>> bytes[0]
b'azerbaijan'
>>> i=bytes
>>> i[0]
b'azerbaijan'
>>> i[1]
'ascii'
>>> i[0]=105
Traceback (most recent call last):
  File "<pyshell#93>", line 1, in <module>
    i[0]=105
TypeError: 'tuple' object does not support item assignment
>>>
```

Əgər bunu bayt massivi ilə etsək xəta ilə rastlaşmayacağıq

```
>>> i=bytearray('azerbaijan','ascii')
>>> i
bytearray(b'azerbaijan')
>>> i[0]=65
>>> i
bytearray(b'Azerbaijan')
>>> i[0]=105
>>> i
bytearray(b'izerbaijan')
>>>
```

Və rahatlıqla azərbaycan sözünün ilk hərfini A və i hərfləri ilə dəyişə bildik

callable() funksiyası

Bu funksiya digər python funksiyalarının çağrıla bilən olduğunu təyin edərək True əks halda False çap edir

```
>>> callable(open)
True
```

```
>>> callable(with)
SyntaxError: invalid syntax
>>> callable(abs)
True
>>> callable(dict)
False
>>> callable(count)
Traceback (most recent call last):
  File "<pyshell#104>", line 1, in <module>
    callable(count)
NameError: name 'count' is not defined
>>>
```

chr() funksiyası

Funksiya Unicode kodlama sistemindəki bazalara əsaslanaraq ədədlərin xarakterik qarşılığını verir

```
>>> chr(34)
'"'
>>> chr(64)
'@'
>>> chr(56)
'8'
>>> chr(65)
'A'
>>>
```

repr()   funksiyası

Bu funksiya ascii() funksiyasından fərqli olaraq,ascii kodlama sistemindən kənarda olan ifadələrlə belə qarşılaşsa ekrana çap edəcəkdir.

```
>>> repr('Qarabağ')
"'Qarabağ'"
```

```
>>> repr('Əlizadə')
"'Əlizadə'"
>>>
```

list() funksiyası

əvvəlki dərslərimizdə listlərin nə olduğu haqqında ətraflı danışmışdıq.Bundan başqa list bir funksiya kimi python3-də yer alır.Boş bir list yaradmaq üçün lst=[] yolundan istifadə edirdiksə,funksiya kimi list,veriləni list-ə çevirməyə yardım edir.

```
>>> i=list()
>>> list()
[]
>>> i
[]
>>> type(i)
<class 'list'>
>>>
```

Yuxarıdakı kodlarımızda i-veriləninə list() funksiyasını əlavə edərək boş bir list yaratdıq.Daha sonra tipini soruşduq və bizə class 'list'-list sinfinə aid olduğunu söylədi.

```
>>> i=list('Alians')
>>> i
['A', 'l', 'i', 'a', 'n', 's']
>>> dict={'mobile':'553235678','name':'Aynur'}
>>> type(dict)
<class 'dict'>
>>> list(dict)
['mobile', 'name']
>>> list
<class 'list'>
>>> i=list(dict)
>>> i
['mobile', 'name']
>>> type(i)
```

```
<class 'list'>
>>>
```

Kodlarımızda bir lüğət yaradaraq daha sonra bu lüğəti list-ə çevirdik.

set() funksiyası

Funksiya,digər siniflərə daxil olan str,list,dict -i çoxluqlara çevirməyə yardımçı olur.

```
>>> i='Angel & Demon'
>>> set(i)
{'e', 'o', 'g', '&', ' ', 'D', 'A', 'm', 'l', 'n'}
>>> lst=['Baki','Academy','Oil']
>>> set(lst)
{'Academy', 'Baki', 'Oil'}
>>>
```

tuple() funksiyası

Bu funksiyada çevirmə işlərini həyata keçirir.Digərlərindən fərqli olaraq tuple() tupllara çevirir.

```
>>> tuple()
()
>>> i='mario'
>>> tuple(i)
('m', 'a', 'r', 'i', 'o')
>>>
```

frozenset() funksiyası

Bu funksiyanı çoxluqlar bəhsində keçmişdik.Burda isə funksiya digər siniflərə daxil olanları dondurulmuş çoxluqlara çevirəcək.

```
>>> lst=['Berlin','Paris','London','Omman']
>>> i=frozenset(lst)
>>> i
frozenset({'Paris', 'London', 'Berlin', 'Omman'})
```

complex() funksiyası

Funksiya ədədlərin həqiqi qiymətdən virtual qiymətinə çevirir.Digər yolla biz virtual qiymətini özümüz də qeyd edə bilərik.Qarışıq ədədlər funksiyası kimi tanınır

```
>>> complex(12)
(12+0j)
```

12 -ədədin həqiqi qiyməti 0-isə virtual qiymətidir.

```
>>> complex(12,2)
(12+2j)
>>>
```

Son kodlarımızda isə 2-qiymətini biz virtual qiymət olaraq özümüz qeyd etdik.

float() funksiyası

Riyaziyyatdan bizə məlumdurki ədədlər tam və kəsirli ədədlər olaraq iki yerə bölünür.Funksiya isə tam ədədi kəsirli ədədə çevirmək üçün istifadə olunur.

```
>>> float(3)
3.0
>>> float(4)
4.0
>>>
```

int() funksiyası

Funksiya ədədi tam ədədə çevirir,eləcədə bir ədədi onluq sayma sistemində qarşılığını göstərir

```
>>> int(12.3)
12
>>> int(12.7)
12
>>> int('12',16)
18
>>> int('10',8)
8
>>>
```

dict() funksiyası

Funksiya lst,str və tuple siniflərini lüğətlərə(dict) çevirir.

```
>>> dict()
{}
>>> i=dict(a=1,b=2,c=3)
>>> i
{'b': 2, 'a': 1, 'c': 3}
>>> lst=(['a',1],['b',2],['c',3])
>>> dict(lst)
{'b': 2, 'a': 1, 'c': 3}
>>>
```

ord() funksiyası

Funksiya veriləni onluq say sistemində qarşılığını verir

```
>>> ord('A')
65
>>> ord('a')
97
>>> ord('v')
118
>>> ord('-')
45
>>>
```

oct() funksiyası

Funksiya verilənin səkkizlik say sistemində qarşılığını verir

```
>>> oct(45)
'0o55'
>>> oct(118)
'0o166'
>>>
```

hex() funksiyası

Funksiya veriləni onaltılıq say sistemində qarşılığını verir

```
>>> hex(12)
'0xc'
>>> hex(23)
'0x17'
```

eval() funksiyası

evaluate sözünün qısaltmasıdır,dilimizə qiymətləndirmək kimi tərcümə olunur.Funksiya riyazi hesablamalarda daha çox istifadə olunur.Yəni funksiya pythonda bir moduldurki hesablama işlərində bizə dəqiq nəticəni verir.Funksiyanın nə işə yaradığına nəzər yetirək

```
i=input('write number:')
c=eval(i)
print(c)
```

```
write number:23/34
0.6764705882352942
>>>
```

Nəticədən göründüyü kimi eval() funksiyası hesablama əməliyyatını rahatlıqla etdi.

eval() funksiyası verilənləri daxilinə yerləşdirə bilməz,xəta ilə qarşılaşacaqsınız

```
>>> eval('i=13')
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    eval('i=13')
  File "<string>", line 1
    i=13
     ^
SyntaxError: invalid syntax
>>>
```

qeyd etdiyimiz kimi eval() daxilində ifadələri ala bilir.

exec() funksiyası

Funksiya vasitəsilə yeni verilənlər bazası yaratmaq olur.Yəni evla() funksiyasından fərqli olaraq funksiya daxilində verilən yerləşdirə bilir.Və bu yeni bir verilən olur.

```
>>> i=4
>>> exec('i=6')
>>> print(i)
6
>>> i
6
>>>
```

Gördüyümüz kimi əvvəl i=4 qeyd etdik daha sonra exec() funksiyası daxilində i-nin qiymətini 6 verərək çapda 6-ədədini əldə etdik.Yəni exec() funksiyası əvvəlki nəticələri silir.

copyright() funksiyası

Funksiya proqramın müəllif hüquqları haqqında əmlumatı əldə etməyimizə yardımçı olur.

```
>>> copyright()
Copyright (c) 2001-2015 Python Software Foundation.
All Rights Reserved.

Copyright (c) 2000 BeOpen.com.
All Rights Reserved.

Copyright (c) 1995-2001 Corporation for National Research Initiatives.
All Rights Reserved.

Copyright (c) 1991-1995 Stichting Mathematisch Centrum, Amsterdam.
All Rights Reserved.
>>>
```

credits() funksiyası

Funksiya bir proqramın hazırlanmasında əməyi keçən şirkət və ya şəxslərə təşəkkür mətnini göstərir.

```
>>> credits()
    Thanks to CWI, CNRI, BeOpen.com, Zope Corporation and a cast of
thousands
    for supporting Python development.  See www.python.org for more
information.
>>>
```

license() funksiyası

Funksiya proqramın lisenziyası haqqında məlumat verir

```
>>> license()
A. HISTORY OF THE SOFTWARE
==========================
Python was created in the early 1990s by Guido van Rossum at Stichting
Mathematisch Centrum (CWI, see http://www.cwi.nl) in the Netherlands
as a successor of a language called ABC.  Guido remains Python's
principal author, although it includes many contributions from others.
.....................................................
All Python releases are Open Source (see http://www.opensource.org for
Hit Return for more, or q (and Return) to quit: q
>>>
```

dir() funksiyası

Funksiya bir növ məlumat bazasıdır.shell-dən

```
>>> dir('')
```

yazsaq string cinsinə daxil olan metodları bizə çap edəcəkdir.

divmod()   funksiyası

Funksiya ədədlərin bölünməsindən əldə olunan cavabı,eləcədə bu bölmə əməliyyatından qalan qalığı çap edir.iki ədəd argument ala bilir.

```
>>> divmod(12,4)
(3, 0)
>>> divmod(26,5)
(5, 1)
>>>
```

Nəticəyə görə 12/4=3 qalan qalıq isə 0-dır
Nəticəyə görə 26/5=5 qalıq isə 1-dir.

divmod() funksiyası qalığı yuvarlaqlaşdırmır.

Enumerate() funksiyası

Funksiya dilimizə listələmə,nömrələmə kimi tərcümə olunur.Funksiya verilənlər daxilində hər bir argumenti nömrələyərək çap edirik.

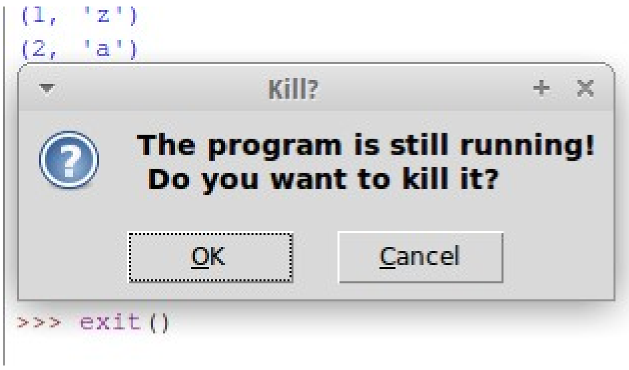
```
>>> dict={}
>>> dict={'a':1,'b':2}
>>> enumerate(dict)
<enumerate object at 0x7fea2020ec60>
```

```
>>> list((enumerate(dict)))
[(0, 'b'), (1, 'a')]
>>>
>>> for i in enumerate('Azadlıq'):
        print(i)


(0, 'A')
(1, 'z')
(2, 'a')
(3, 'd')
(4, 'l')
(5, 'ı')
(6, 'q')
>>>
```

exit() funksiyası

Funksiya dilimizə çıxış,tərk etmək kimi tərcümə olunur.Funksiyanı python shell-də yazsanız həmin an python shell-dən çıxış edəcəksiniz



id() funksiyası

Pythonda bu parametr ifadələrin passport nömrəsini göstərir.İfadənin yaddaşda tutduğu say nömrəsini çap edir.

```
>>> id(40)
9996736
>>> i=24
>>> id(i)
9996224
>>> id(24)
9996224
>>> id('24')
140643569242664
>>>
```

format() funksiyası

format() funksiyası format() metoduna bənzəyir.Funksiyanın əsas xüsusiyyəti, veriləni(cinslərindən asılı olaraq,int,float,str) ekrana çap edir,daha dəqiq desək verilənin cinsinə uyğun olaraq açıq şəkildə göstərir.

```
>>> format(2,'2o')
' 2'
>>> format(2,'f')
'2.000000'
>>> '{:f}'.format(3)
'3.000000'
>>>
```

hash() funksiyası

Funksiya veriləni qarışıq şəkildə bir nəticəsini tap say olmaq şərti ilə ekrana çap edir.

```
>>> hash('ellipse')
1718871312798335030
>>> hash('code')
4894350872844396189
>>> hash('1')
-3712752847035930642
>>>
```

isinstance() funksiyası

Funksiya type() funksiyasına bənzəyir.Amma bu funksiya argument alaraq verilənin cinsini True və ya False ifadələri ilə bizə doğru ya olmdadığını nəticə etibarı ilə göstərir.Misallara baxaq.

```
>>> isinstance('прогулка',str)
True
>>> isinstance(23,str)
False
>>> isinstance(23,int)
True
>>>
```

Nəticələrdən aydın olduğu kimi biz sözlə ifadə etməyə çalışaq.İlk 'прогулка' yazaraq cinsinin str olduğunu soruşduq və bizə True(yəni doğru) verdi.İkinci kodumuzda funksiyadan istifadə edərək 23-ədədinin str cinsinə aid olduğunu soruşduq və bizə False verdi(yəni səhv,yanlış)Bəli bu doğrudur,23-ədədi tam ədəddir və cinsi bildiyimiz kimi int(integer)-dir.Son olaraq int-ifadəsi ilə soruşduq və True aldıq.

len()   funksiyası

Funksiyanı bir də təkrar edərək misallar yazaq.

```
>>> i=[1,2,3,4,5,6,7,8,9]
>>> len(i)
9
>>> len('python')
6
>>>
```

max() funksiyası

funksiya verilənlərin ən böyük dəyərə malik olanı çap edir.Bir argument ala bilir.

```
>>> i=[1,2,3,4,5,6]
>>> max(i)
6
>>> i=[3,1,6,9,5,3,0,7]
>>> max(i)
9
```

```
>>> i=['Vahid','Kəmaləddin','Səmayə','Xəyalə']
>>> max(i,key=len)
'Kəmaləddin'
>>
```

min() funksiyası

Funksiya max() funksiyasının tərsini edərək,verilənlər arasında ən kiçik ifadəni çap edir.

```
>>> i=['Kəmaləddin','Xəyalə','Arzuman']
>>> min(i,key=len)
'Xəyalə'
```

```
>>>
```

open()   funksiyası

Funksiya ilə əvvəldən tanışıq.Funksiya bir neçə argument alaraq yeni fayl yaratmaq üçün,eləcədə var olan faylı oxumaq üçün istifadə oluna bilən pythonda kök funksiyalardandır.

```
>>> f=open('techazweb.rb','a')
>>> f=open('techazweb.rb','r')
>>> f.read()
''
>>>
```

xüsusi işarələr fayl adı əvvəlində yazılarsa xəta alacağınızı bildiyiniz üçün təkrar edərək r-hərfini başa əlavə edək

```
>>> f=open('c:\night.txt','a')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
OSError: [Errno 22] Invalid argument: 'c:\night.txt'
```

və xəta aldıq

```
>>> f=open(r'c:\night.txt','a')
>>>
```

aşağı sətrə səhvsiz keçid etdik

buffering argumenti(parametr)

buffering parametri əsasən yaratdığımız fayl daxilinə ifadələri sətir-sətir əlavə etməyə kömək edir.

```
>>> f=open('Mango.txt','w',buffering=1)
>>> f.write('system\n')
7
>>> f.write('texno')
5
```

```
>>> f.close()
>>>
```

və faylımıza daxil olduqda

system
texno

yazılarını görə bilərik

fayllarımızı əvvəlcədən dil kodlamasını tətbiq edərək

```
>>> f=open('Mango.txt','w',buffering=1,encoding='utf-8')
>>>
```

pow() funksiyası

Dilimizə qüvvət kimi tərcümə olunur.ədədi qüvvətə yüksəldir.toplam üç sayda argument ala bilir.

```
>>> pow(3,2)
9
>>> pow(3,2,2)
1
>>>
```

Son nəticəyə isə biraz izahat gətirərək 3 ədədinin 2-yə qüvvətə yüksəltməsindən 9-ədədini əldə edirik,daha sonra yazdığımız son 2,9-ədədini ikiyə bölərək qalığı ekrana çap etdirdik.

```
     1  2   3
     |  |   |
pow( 3,  2,  2)
```

1-qüvvətə yüksəldəcəyimiz ədəd
2-qüvvət yüksəltməsi

3-alınan cavabın 2-yə bölünməsi

reversed() funksiyası

Dilimizə tərs çevirmə kimi tərcümə olunur.Pythonda da bu funksiyanı yerinə yetirir.

```
>>> i='ship','mind','shop','bazarstore','safastore'
>>> reversed(i)
<reversed object at 0x7f49b8377e48>
```

reversed() funksiyası ekrana təyini çap üçün list funksiyasından istifadə edirik.

```
>>> list(reversed(i))
['safastore', 'bazarstore', 'shop', 'mind', 'ship']
>>>
```

sorted()   funksiyası

Funksiya ifadələri əlifba sırası ilə düzür.

```
>>> sorted(i)
['bazarstore', 'mind', 'safastore', 'ship', 'shop']
>>> sorted('python')
['h', 'n', 'o', 'p', 't', 'y']
>>>
```

slice() funksiyası

Funksiya verilənlər daxilində bölmə işini yerinə yetirir.

```
>>> i=['Vahid','Eldar','Sahil','Mətanət']
>>> slice(i)
slice(None, ['Vahid', 'Eldar', 'Sahil', 'Mətanət'], None)
>>> v=slice(0,2)
>>> i[v]
['Vahid', 'Eldar']
>>>
```

sum() funksiyası

Funksiya ədədlərin toplamını verir.iki sayda argument ala bilir.

```
>>> i=[1,2,3,4,5]
>>> sum(i)
15
>>> sum(i,5)
20
```

Ikinci aldığı argument(5) isə i-daxilindəki ədədlərin cəminə +5-ədədini əlavə edir.

lambda funksiyası

Funksiya iki argument alaraq istənilən riyazi hesablamanı yerinə yetirir.Bir növ qısa yoldan hesablama üsuludur.

```
>>> lambda a,b: a**2-b*1 (4,5)
<function <lambda> at 0x7fdda8bdb510>
```

```
>>> i=lambda a,b: a**2-b*1
>>> i(4,5)
11
```

Son kodlarımızda, lambda funksiyasını bir dəyişənə atdıq(i) və veriləni 4 , 5 olaraq qeyd edib cavabı əldə etdik.Yəni

4**2=16 b*1=5*1=5
16-5=11

lambda,daxilində digər kök funksiyaları da ala bilir

```
>>> i=lambda a: pow(a,3)
>>> i(3)
27
>>>
```

Funksiyalar

Əvvəlki bölmələrdə bəhs etdiyimiz funksiyalar pythonda kök funksiyalar olub,python developerlər(python-u inkişaf edənlər) tərəfindən hazır yazılmış funksiyalardır.Və biz istifadəsində xəta almadan rahatlıqla istifadə edə bilirik.Bu kök funksiyalardan başqa bunları əvəz edə biləcək ,bizim ideyalarımıza dayana bilən alternativ yollarda mövcuddur. Kök funksiyalara misal olaraq print() open() str() və s misal çəkmək olar.Bunlardan başqa biz yarada biləcəyimiz metodlar da var.Bu metod def adlanan ifadədən başlayır.Yəni bir funksiyanı yaratmaq üçün def

ifadəsi köməyimizə çatacaq

def funksiya_adı(): ifadəsində iki-nöqtədən sonra aşağı bir tab düyməsi qədər keçid edərək funksiyamızın kökünü yarada bilərik.Yaradacağımız funksiya daxilindəki kodlar bəsit deyil,istifadəçilərə,eləcədə bizim işimizə yaraya bilən kodlardan təşkil olunmalıdır.Proqramlaşdırmada məqsəd görünüş deyil,əsas yazdığınız kodların bir işi xətasız yerinə yetirməsidir.İstər proqramınızı qrafik(GUİ) yada terminaldan istifadə edəcək şəkildə yazın nəticə etibarı ilə bir işə yaramırsa o artıq biz script deyil.

```
>>> pow(2,3)
8
>>>
```

Yuxarıda istifadə etdiyimiz pow() python-da developerlər tərəfindən kök funksiyadır və qüvvətə yüksəltməni icra edir.Əgər bunu başqa yolla əvəz etsək bilirikki

```
>>> 2**3
8
>>>
```

kimi olacaq,Yəni iki ulduz işarəsi köməyimizə çatdı.Əsas məsələ bu deyil.Məsələ ondadırki biz hər dəfə python-shell açaraq ədədləri daxil edib yorucu işlə məşğul olacağıq.Buna görədə python bizə def ifadəsi ilə bir alternativlik verir

İlk qüvvətə yüksəltmə funksiyasını yazaq

```
def ilk_funksiya():
    i=int(input('ədəd daxil edin:'))#qüvvətə yüksəldəcəyimiz ədəd üçün
    v=int(input('ikinci ədədi daxil edin:'))# qüvvət yüksəltməsi üçün ədəd
    print('cavab=',pow(i,v))
ilk_funksiya()
```

Yuxarıdakı kodlarımızı bir fayla yazıb sonunu ilk_funksiya.py(windows istifadəçiləri pyw olaraq qeyd etsin) yazaraq yaddaşa veririk.

Daha sonra terminalı açırıq python3 scriptinin olduğu qovluğa daxil oluruq

```
$ cd Desktop
:~/Desktop$ ls
arduino.desktop  ilk_funksiya.py  operation       pyfirm.pyc      yeni.cpp
Dev-C++.desktop  include          os              python3-az.odt  yeni.o
fato.py          library          program_run.sh  python3-az.pdf
fato.pyc         New File         program_run.sh~  yeni.c
:~/Desktop$ chmod +x ilk_funksiya.py
:~/Desktop$ python3 ilk_funksiya.py
ədəd daxil edin:3
ikinci ədədi daxil edin:3
cavab= 27
:~/Desktop$
```

Və gördüyünüz kimi rahatlıqla terminaldan çalışdıraraq istifadə edə bildik Funksiyalarda hətda müəyyən parametrlər daxil edərək də istifadə edə bilərik.

```
def country(country,city,avenu,home):
    print('country:',country)
    print('city:',city)
    print('avenu:',avenu)
    print('home:',home)
country('Azerbaijan','Ganja','H.Zardabi','ev/1')
```

```
>>>
country: Azerbaijan
city: Ganja
avenu: H.Zardabi
home: ev/1
>>>
```

Yuxarıdakı kodlarımızda country() funksiyası daxilində təyinatlı parametrlər verərək və son funksiyanı bağlayarkən təyinatla ardıcıl olaraq ifadələri daxil etdik.

```
def os():
    import os
    if os.name=='posix':
        print('Hello linux user')
    elif os.name=='nt':
        print('Hello windows user')
    else:
        print('Unknown os x type')
os()
```

```
>>>
Hello linux user
>>>
```

Kodlarımız içində os() funksiyası yaratdıq və funksiyanın nə işə yarayacağı üçün import os-yəni os modulunu çağırdıq və ardından if operatoru vasitəsilə əməliyyat sisteminin posix olduğu zaman ekrana print() funksiyası vasitəsilə 'Hello linux user' yazdıq.və elif else operatorları da bizə yad deyil.Deməli funksiyalar daxilində modullardan da istifadə edə bilərik.Amma əgər modulu funksiyanın daxilində (yuxarıdakı kodlarımız kimi) yazarsaq növbəti funksiyada os moduluna ehtiyac olanda xəta ilə qarşılaşacağıq.Mən bunu istifadə edə biləcəyimiz üçün yazdım.

Əgər yadınızdadırsa cüt və tək ədədlərin təyini ilə bağlı belə bir kod yazmışdıq.

```
while True:
    i=int(input('ədəd yazın:'))
    if i%2==0:
        print('{} ədədi cüt ədəddir'.format(i))
    else:
        print('{} ədədi tək ədəddir'.format(i))
```

```
>>>
```

ədəd yazın:23
23 ədədi tək ədəddir
ədəd yazın:4
4 ədədi cüt ədəddir
ədəd yazın:6
6 ədədi cüt ədəddir

Və kodlarımız xətasız çalışır.Bu kodlarımızı funksiya daxilinə yerləşdirək

```
def eded():
    while True:
        i=int(input('ədəd yazın:'))
        if i%2==0:
            print('{} ədədi cüt ədəddir'.format(i))
        else:
            print('{} ədədi tək ədəddir'.format(i))
eded()
```

```
>>>
ədəd yazın:23
23 ədədi tək ədəddir
ədəd yazın:34
34 ədədi cüt ədəddir
ədəd yazın:1
1 ədədi tək ədəddir
ədəd yazın:
```

Funksiyalarda istənilən an parmetrlərimizi dəyişib fərqli nəticələr ala bilərik.Misallara baxaq

```
def fayl(i='/home/user/'):
    f=open(i+'fayl.txt','a')
    print('fayl {} qovluq altında quruldu'.format(i))
fayl()
```

və ekran çapı

```
>>>
fayl /home/user/ qovluq altında quruldu
>>>
```

kimi olacaq.
İlk əvvəl qovluğun yerini göstərdik və sonra fayl.txt adlı fayl yaratdıq və sonda fayl() funksiyasını bağladıq.

Biz qovluğu dəyişmək üçün hər dəfə python scriptimizi açaraq i='/home/user/Desktop/' yazmaq əvəzinə

```
>>> fayl('/home/user/Desktop/')
fayl /home/user/Desktop/ qovluq altında quruldu
>>>
```

fayl('/home/panda/Desktop/') yazaraq işimizi asanlaşdıra bilərik.Gördüyünüz kimi funksiyalar vasitəsilə nəqədər işimiz yüngülləşir.

Əgər fayl() boş yazsaq funksiya əvvəlki parametrdən istifadə edərək '/home/user/ qovluğu altında faylımızı yaradacaq

```
>>> fayl()
fayl /home/user/ qovluq altında quruldu
>>>
```

```
def vurma(*ededler):
    i=1
    for v in ededler:
        i*=v
    print(i)
vurma(2,3,4)
```

```
>>>
24
>>>
```

Yuxarıdakı funksiyamız necə çalışdığına dair açıqlıq verək

*ededler-ifadəsində *-işarəsi vurmanı təmsil edir.
Daha sonra i=1 qeyd edirik.
i*=v ifadəsi i*v ilə eynilik təşkil edirik.
Və ədədlərimiz 2,3,4

və sonra *ededler ifadəsi vurma(2,3,4) ilə əvəz olunaraq

1*(2*3*4)=24 nəticəsin verəcək

Əgər

i=1 -ifadəsini i=2 ilə əvəz etsək

```
def vurma(*ededler):
    i=2
    for v in ededler:
        i*=v
    print(i)
vurma(2,3,4)

>>>
48
>>>
```

yəni

i=2 olduğundan 2*(2*3*4)=48 nəticəsini aldıq

Hər dəfə

```
>>> vurma(1,2)
4
>>> vurma(1,2,3,4)
48
>>>
```

vurma funksiyasına fərqli parametrlər verərək nəticəni əldə edə bilərik.

Funksiyalar bizə daha çox kod blokları yazmağa yardım edəcək bir funksiyadır.

global dəyişəni (variable)

global dəyişəni,funksiyadan kənarda verilən parametrləri funksiya daxilində funksionallığını təmin edir.Aşağıdakı misala baxaq

```
list=[]
def globals():
    list.append('batlle')
    print(list)
globals()
```

kodlarımızı çalışdırdıqda boş bir nəticə əldə edəcəyik,çünki list funksiyadan kənardadır.Bunun üçün global dəyişənindən istifadə edəcəyik

```
list=[]
def globals():
    global list
    list.append('batlle')
globals()
print(list)
```

```
>>>
['batlle']
>>>
```

və funksiyadan kənarda da print(list) yazaraq istədiyimiz nəticəni əldə etdik.
Funksiya ilk əvvəl def funk()   daxilindəki kodları oxuyur və nəticəni

```
                              .............
                              funk()
```

aralıqda tapmayınca global ifadəni-kənar qiymətləri oxuyur

```
i=0
def funk():
    print(i)
funk()

>>>
0
>>>
```

Gördüyümüz kimi funksiyadan kənarda i=0 yazdıq,amma funksiya daxilində verilən təyin etmədik.Və funksiya ilk daxilindəki kodları oxudu bir şey tapmayınca global ifadəmiz i=0 dəyərini ekrana çap etdi

əgər

```
i=0
def funk():
    i=10
    print(i)
funk()
```

kimi yazsaq

```
>>>
10
>>>
```

nəticəsini əldə edərik.Və yuxarıda qeyd etdiyim kimi funksiyadan kənarda i-nin dəyərini oxumadı.global dəyişənindən istifadə edərək

```
i=0
global i
def funk():

    i=10
    print(i)
funk()
```

```
print(i)
```

yaza bilərik
və ekrana həm funksiya daxilindəki i-nin qiymətini həmdə funksiyadan kənarda i-nin qiymətini ekrana çap edə bilərik

```
>>>
10       #funksiya daxilindəki    i=10
0        #funksiyadan kənarda  i=0
>>>
```

global dəyişənini harda qeyd etsənin ordakı global ifadənin dəyərini qeyd edəcək.

Funksiyaları hətda operatorla bərabər çağıra bilərik

```
def pair():
    print('{} ədədi təkdir'.format(i))
def odd():
    print('{} ədədi cütdür'.format(i))
i=input('ədədi daxil edin:')
if int(i)%2==0:
    odd()
else:
    pair()
```

```
>>>
ədədi daxil edin:5
5 ədədi təkdir
>>>
```

Qeyd edimki funksiyalar nəqədrki açıq qalırsa,funksiya çalışmağa hazır deyil,o vaxta qədərki biz funksiyanı bağlayaq.

```
def name(ad):
    print('Salam %s'%ad)
name('Rashad')

>>>
```

```
Salam Rashad
>>>
```

```
def vurma(list):
    i=1
    for v in list:
        i=i*v
    print(i)
list=[1,2,3]
vurma(list)
```

```
>>>
6
>>>
```

```
def vurma(list):
    i=2
    for v in list:
        i=i*v
    print(i)
list=[1,2,3]
vurma(list)
```

```
>>>
12
>>>
```

return və pass operatorları(təkrar)

return operatoru dilimizə geri dönmək,qayıtmaq kimi tərcümə olunur. Operator verdiyimiz parametrə geri dönərək təkrar verilənin dəyərinə baxır

```
def oper():
    i=12
    print(i)
print('i-nin qiyməti- %s'%oper())
```

və ekran çapında

```
>>>
12
i-nin qiyməti- None
>>>
```

nəticəsini əldə etdik.

Nəticədə None deyə ifadə əldə etdik,bunun əksinə iki print() funksiyasından istifadə edərək hər iki halda 12 qiymətini çap etdirmək istəsək də buna nail olmadıq.
Burada return operatorundan istifadə edərək növbəti print() funksiyasına da i-nin qiymətini əlavə edək

```
def oper():
    i=12
    print(i)
    return i
print('i-nin qiyməti- %s'%oper())

>>>
12
i-nin qiyməti- 12
>>>
```

və gördüyümüz kimi None-əvəzinə istədiyimiz nəticəni əldə etdik.

```
def ret(passw):
    if len(passw)<1:
        pass

    else:
        print(passw)
        return ret(passw[1:])
print(ret('kombobox'))
>>>
kombobox
```

```
ombobox
mbobox
bobox
obox
box
ox
x
None
>>>
```

```
def ret(passw):
    if len(passw)<1:
        return passw

    else:
        print(passw)
        return ret(passw[1:])
print(ret('kombobox'))
```

```
>>>
kombobox
ombobox
mbobox
bobox
obox
box
ox
x

>>>
```

Əgər bir list olaraq çap etdirmək istəsək

```
def ret(i):
    if len(i)<1:
        return i
    else:
        print(list(i))
```

```
        return ret(i[1:])
ret('system')

>>>
['s', 'y', 's', 't', 'e', 'm']
['y', 's', 't', 'e', 'm']
['s', 't', 'e', 'm']
['t', 'e', 'm']
['e', 'm']
['m']
>>>
```

pass operatoru

Operator dilimizə ötürmək kimi tərcümə olunur.Funksiyası da veriləni nəzərə almadan ötüb keçməkdir.

```
list=[]
i=input('write something:')
list.append(i)
print(*list)
if i not in list:
    pass

>>>
write something:

>>>
```

Yuxarıda gördüyümüz kimi əgər I- list daxilində yoxdursa pass-yəni ötüb keçəcək

```
write something:tryu
tryu
>>>
```

Funksiya daxilindədə həmçinin

```
>>> def hello():
	pass

>>>
```

Tənliyin köklərini tapaq,eləcədə pass operatorundan istifadə edək.

```
def diskrim():
    try:
        a=float(input('a --> '))
        b=float(input('b --> '))
        c=float(input('c --> '))
        d=(b**2)-4*a*c
        if d>0:
            k1=(-b+(d**2))/(2*a)
            k2=(-b-(d**2))/(2*a)
            print('k1={}'.format(k1))
            print('k2={}'.format(k2))
        elif d==0:
            k1=-b/(2*a)
            print('k1={}'.format(k1))
        else:
            print('[-] No root of the equation')

    except (ValueError,NameError):
        print('[-]')
        pass
diskrim()
```

```
>>>
a --> 1.8
b --> 1.2
c --> -1.4
```

```
k1=36.53066666666667
k2=-37.197333333333326
>>>

>>>
a --> d
[-]
>>>
```

```
base=[]
global base
def say():
    name=input('write your name:')
    surname=input('write your surname:')
    mobile=input('write you mobile number:')
    base=[name,surname,mobile]
    print('database  : --> %s <--'%base)
    print('name       : %s '%name)
    print('surname  : %s '%surname)
    print('mobile     : %s '%mobile)
say()
```

```
>>>
write your name:Ramin
write your surname:Mamedov
write you mobile number:+7034567891
database  : --> ['Ramin', 'Mamedov', '+7034567891'] <--
name       : Ramin
surname  : Mamedov
mobile     : +7034567891
>>>
```

len() funksiyasının əvəzedicisi kimi

```
def lenth():
```

```
    ques=input('ifadə yazın : ')
    i=0;
    for v in ques:
        i +=1;
    print(i);
lenth()
```

```
>>>
ifadə yazın : alabama
7
>>>
```

və kodlarımızı python skripti olaraq lenth.py qeyd edib terminaldan hər an çalışdıra bilərik

Funksiyalarda args və kwargs argumentləri

Bu iki arqumentlər ulduz işarələri ilə bərabər işlədilir.Məqsədi funksiya daxilində bir və ya bir neçə ifadəni çap etdirməkdir.Bir tərəfdən print() funksiyasının rolunu icra edir.
Misallara baxaq

*args və **kwargs arqumentləri,args lüğətlərin daxilində açar dəyərlərini taparaq çap edirik.

```
def arg(*arg,**kwargs):

    for dict in kwargs:
        if dict:
            print('{}'.format(dict))
        else:
            print('[-]  {}'.format(dict))
dict={'program':'python','platform':'linux','version':'3.4.5'}
arg(**dict)
```

```
>>>
```

```
program
platform
version
>>>
```

```
def arg(**args):
    print(args)
arg(proqram='Python',version='3.4.5',platform='Linux')

>>>
{'platform': 'Linux', 'version': '3.4.5', 'proqram': 'Python'}
>>>
```

Və yerdəyişmə olmadan yazdığımız ifadələr çap olundu.args parametri əvəzinə siz istənilən ifadə yaza bilərsiniz.Sadəcə olaraq args yazmaqda məqsəd,kodlarımızı hər kəsin başa düşəcəyi şəkildə ifadə etməkdir.

əgər

```
def arg(**sol):
    print(sol)
arg(proqram='Python',version='3.4.5',platform='Linux')
```

kimi yazsaq,xətasız ekran çapına nail olacağıq.

Bu argumentllər,print() funksiyasında sep() metodunu əvəz edərək

```
def arg(*args,i='',**kwargs):
    for man in args:
        print(i+man,**kwargs)
arg('Adil','Fazil','Sahil',i='!.')

>>>
!.Adil
!.Fazil
!.Sahil
```

```
>>>
```

verilənlərin önünə ifadələr əlavə edə bilir.

Modullar(modules)

Python daxilində standart modullarla gəlir.Bu nöqteyi-nəzərdən modulları iki yerə bölə bilərik

1.Hazır modullar
2.Bizim yarada biləcəyimiz modullar

1-ci şərtə daxil olan modullar python-u yüklədikdə,onunla bərabər gəlir. Biz bundan əvvəlki bölmədə funksiyalardan danışdıq.Funksiyaların bizim üçün nəqədər əlverişli şərait yaratdığının şahidi olduq.Modullar da həmçinin yazacağımız kodları bir yerə toplayaraq ayrı-ayrı yerlərdən çağırılaraq istifadəsinə təminat verir.Həmçinin modullardan istifadə edərək funksional skriptlər yaza bilərik.Standart modullar hər biri bir vəzifəni icra edir.urllib,os,sys modulu və sairə.Pythonda standar kitabxana modulları haqqında https://docs.python.org/3/library/ ünvandan ətraflı məlumat ala bilərsiniz.

İndi də gəlin modulu çağırma metoduna baxaq

import modul_adı

Yuxarıdakı metoddan istifadə edərək bir modulu çağıraq.

```
>>> import os
>>>
```

və aşağı sətrə xətasız keçid etdik.Deməli modul bizdə yüklüdür.Standart

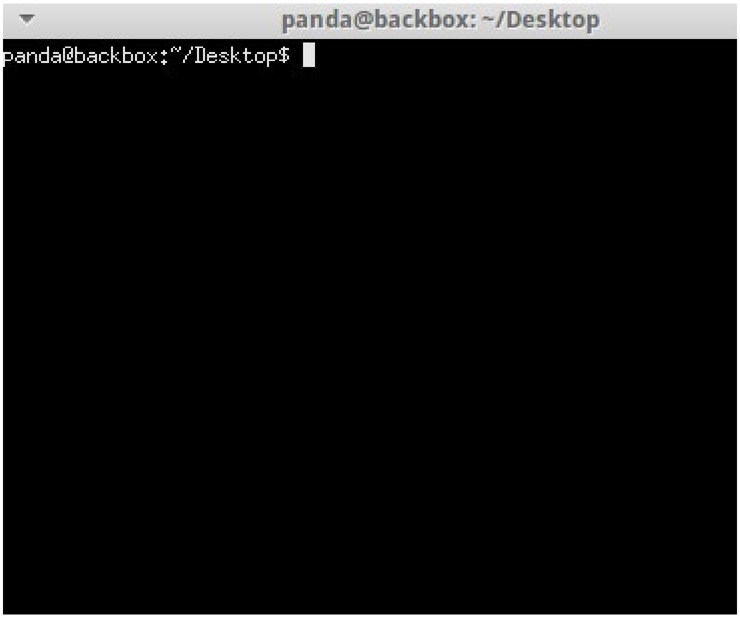
modullar python qovluğu ilə bərabər yükləndiyindən xəta vermə ehtimalı yoxdur.
Və dir funksiyasını keçdiyimiz üçün

```
>>> dir(os)
```

yazaraq qarşımıza çıxan metod və funksiyalara nəzər yetiririk.
Və onlar içindən system() funksiyasından istifadə edək

```
>>> import os
>>> os.system('xterm')
```

və qarşımıza terminal çıxdı



İlk əvvəl import os yazaraq os modulunu çağırdıq ardından os modulu daxilində olan system() funksiyasını xterm parametri ilə çağıraraq(os.system('xterm')) moduldan istifadə edə bildik.

Bu bölmədə biz modullardan istifadə qaydalarına ümumi baxacağıq,daha sonra bir neçə moduldan istifadə edərək skriptlər yazacağıq.
Haşiyədən kənara çıxmayaraq os modulunun içindəki name metodunu çağıraq

```
def sysinfo():
    import os
    if os.name=='posix':
        print('Hello linux user')
    elif os.name=='nt':
        print('hello windows user')
    else:
        print('unknown type Osx')
sysinfo()

>>>
Hello linux user
>>>
```

Kodlarımızı izah edək:

ilk əvvəl funksiya yaratdıq,daha sonra os modulunu çağırdıq və ardından əməliyyat sistemlərini sorğuya və
print('Hello linux user') ifadəsi vasitəsilə istifadəçiyə salam göndərdik.Qeyd edimki posix nt və s əməliyyat sistemlərinin təyini argumentləridir.
Ətraflı olaraq

"posix" -gnu/linux
"nt", "dos", "ce" - Windows
"mac" -Macintosh
"os2" – os/2

....

digər yolu

```
>>> import os
>>> os.name
'posix'
>>>
```

Yazaraq əməliyyat sisteminizi təyin edə bilərsiniz

import modul_adı as

Bu yolla modul daxilində olan funksiyalardan rahatlıqla istifadə etmək olar.Bir növ faylları açarkən istifadə etdiyimiz yola bənzəyir

```
>>> import os
>>> import os as i
>>> i.name
'posix'
>>>
```

Gördüyünüz kimi os modulunu bir verilənə atdıq(i) və bu verilənlə name metodunu istifadə etdik.

Həmçinin

```
>>> import subprocess as i
>>> i.call('gedit')
```

yazaraq sistemdəki gedit proqramını çağıra bilərik.Çağırdığınız proqram yüklü olmalıdır,əgər yüklü deyilsə

```
>>> import subprocess as i
>>> i.call('dumpper.exe')
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    i.call('dumpper.exe')
  File "/usr/lib/python3.4/subprocess.py", line 537, in call
    with Popen(*popenargs, **kwargs) as p:
  File "/usr/lib/python3.4/subprocess.py", line 859, in __init__
    restore_signals, start_new_session)
  File "/usr/lib/python3.4/subprocess.py", line 1457, in _execute_child
    raise child_exception_type(errno_num, err_msg)
FileNotFoundError: [Errno 2] No such file or directory: 'dumpper.exe'
>>>
```

xəta alacağsınız.

Və ya

```
>>> import webbrowser as i
>>> i.open('http://www.python.org')
True
>>>
```

www.python.org ünvanı browser-lə avtomatik qarşımıza çıxdı.

Modul daxilindən import edərək bir neçə funksiya və ya metodları da bu yolla çağıra bilərik.
from modul_adı import metod və funksiyalar

Bu yolla həm bütünlükdə funksiya və metodları çağırmaq olar həm də tək bir metod və ya funksiyanı
Yəni

from os import* -bu yol os modulunun içindəki istənilən funksiya və metodu istifadə etməyimizə kömək edir.

from os import name,system və s -yolu isə sadəcə name və system metod və funksiyasını istifadə etməyə kömək edir.

from os import name-açıqlaması isə,os modulundan sadəcə name metodunu çağır

```
>>> from os import name
>>> name
'posix'
>>>
```

Bu yolla sadəcə name yazmaqla sistemimiz haqqında məlumat əldə etdik

Əgər

```
>>> system('xterm')
Traceback (most recent call last):
  File "<pyshell#47>", line 1, in <module>
    system('xterm')
NameError: name 'system' is not defined
>>>
```

yazsaq xəta ilə qarşılaşacağıq.Çünki biz sadəcə name metodunu çağırmışıq
Amma

```
>>> from os import name,system
>>> system('xterm')
0
>>> name
'posix'
>>>
```

yazsaq hər iki metod və funksiyadan istifadə edə bilərik.

Bütünlükdə funksiya və metodların istifadəsi üçün

```
>>> from os import*
>>>
```

yolundan istifadə etmək lazımdır.Bu yolu amma bir funksiya daxilində çağırsaq xəta alacağıq,odurki bunu funksiyadan xaricdə,global dəyər olaraq qeyd etmənizi məsləhət görürəm.

```
def os():
    from os import*
    if name=='posix':
        print('hello')
    else:
        print('nothing')
os()
```

və funksiyamız çalışmayacaq.
Amma

```
from os import*
def os():
    if name=='posix':
```

```
        print('hello')
    else:
        print('nothing')
os()



>>>
hello
>>>
```

funksiyadan əvvəl yazsaq xəta vermədən,hətda name metodunu tək istifadə edərək ekrana çap etdik.

Bundan başqa as yolu da xəta verir funksiya daxilində.Yəni

```
def os():
     import os as i
...........................
```

Bu yoldan istifadə etək üçün pass operatorundan istifadə etmək olar.Amma təklif edirəmki modulu global olaraq ən başdan çağırmağınız ən alternativ variantdır.

```
def os():
    pass
from os import*
if name=='posix':
    print('hello')
else:
    print('nothing')
os()



>>>
hello
>>>
```

Modulların hazırlanılması

Bizim yaza biləcəyimiz modullar bu standart modullardan heç də fərqlənmir.Çünki bu standart modulların bəziləri də bir py sonluqlu fayl

olaraq pythonxxx qovluğunda yerləşir.Buna əmin olmaq üçün

```
>>> import os
>>> os.__file__
'/usr/lib/python3.4/os.py'
>>> import tkinter
>>> tkinter.__file__
'/usr/lib/python3.4/tkinter/__init__.py'
>>>
```

və python qovluğu altında həqiqətəndə py sonluqlu bir fayldır.Qeyd edimki bu sorğunu yalnız modul daxilində __file__ metodu olanlara baxa bilərik.

```
>>> math.__file__
Traceback (most recent call last):
  File "<pyshell#57>", line 1, in <module>
    math.__file__
AttributeError: 'module' object has no attribute '__file__'
```

və xəta ilə rastlaşdıq.Çünki python modullarının bəziləri python ilə yazılıb,digərləri isə c və c++ dillərindən istifadə edildiyi üçün biz __file__ metodu ilə onları çağıranda xəta ala bilərik.

Bura qədər yazdığımız bir çox py sonluqlu skriplər də deməli bir moduldur.

Modullarımızı çağırma yolları

Necə funksiyaları çağırırdıqsa modullarıda çağırmağın yolları var.
Qeyd edəkki əgər yazdığınız proqramı modul kimi çağırmaq istəyirsinizsə,skriptiniz pythonu açdığınız qovluqda olmalıdır,əks halda No module name filan kimi xəta alacaqsınız.Modulu çağırarkən python ilk əvvəl pythonun olduğu path -yolunda axtarır daha sonra pythonu açdığınız qovluğa nəzər yetirir.Və bu iki yerdə tapmayınca bizə No module name filan xətasını verir.

```
print("""
+++++++++++++++
+  Program running        +
+                            +
+                            +
+                            +
+++++++++++++++
""")
import sys
def file():
    i=input('write key : ')
    v=input('file name:')
    k=input('write file mode : ')
    if i=='start':
        print('{} file opened'.format(v))
        f=open(v+'.txt',k)
    else:
        print('[-] unknown key...')

file()
```

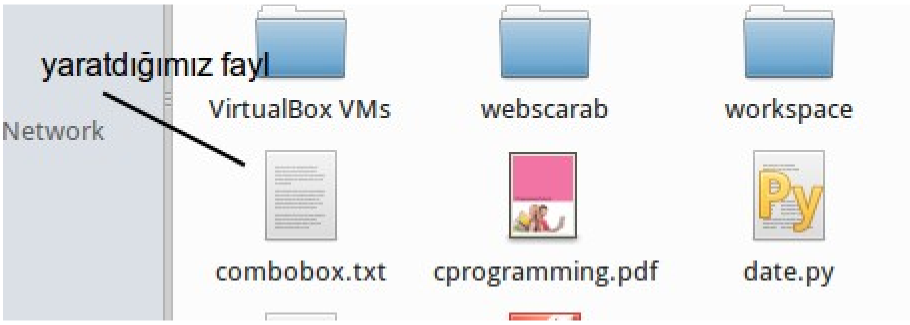
Kodlarımızı date.py yazaraq yaddaşa veririk.(siz nə ad vermək istəsəz verə bilərsiniz)
Daha sonra import date modulunu çağırırıq

```
>>> import date

++++++++++++++
+  Program running   +
+                        +
+                        +
+                        +
++++++++++++++


write key : start
```

file name:combobox
write file mode : a
combobox file opened
>>>



və qovluğumuza baxıb faylın açıldığına əmin olduq.Təkrar qeyd edirəmki py faylınız pythonun açıldığı qovluqda olmalıdır yoxsa modulu çağırdıqda xəta alacaqsınız.

Və ya python PATH yolu ilə pythonu çalışdırdığımız yolu kökə əlavə edərək istənilən yazdığımız modulu hər zaman rahat istifadə edə bilərik

```
>>> import sys
>>> sys.path
['', '/home/panda', '/usr/bin', '/usr/lib/python3.4', '/usr/lib/python3.4/plat-x86_64-linux-gnu', '/usr/lib/python3.4/lib-dynload', '/usr/local/lib/python3.4/dist-packages', '/usr/lib/python3/dist-packages']
>>> sys.path.append(r'/home/user')
>>> sys.path
['', '/home/user', '/usr/bin', '/usr/lib/python3.4', '/usr/lib/python3.4/plat-x86_64-linux-gnu', '/usr/lib/python3.4/lib-dynload', '/usr/local/lib/python3.4/dist-packages', '/usr/lib/python3/dist-packages', '/home/panda']
>>>
```

home-user qovlu]umuzu path-a əlavə etdik.

Date.py skriptini açırıq,sonda yazdığımız file() ifadəsini silirik.və

```
>>> import date
```

```
>>> dir(date)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'start', 'sys']
>>>
```

yazaraq istifadə edə biləcəyimiz funksiyalara baxırıq.
Gördüyünüz kimi list daxilində start adlı istifadə edəcəyimiz bir funksiya var.Əvvəl start funksiyasından başqa,digərlərinə baxaq

```
>>> date.__name__
'date'
```

__name__ funksiyası modulumuzun adını verdi

```
>>> date.__file__
'/home/panda/date.py'
```

__file__ isə qovluğumuzun yerini göstərdi

```
>>> date.__spec__
ModuleSpec(name='date', loader=<_frozen_importlib.SourceFileLoader
object at 0x7f7482b3e860>, origin='/home/panda/date.py')
>>> date.__package__
''
>>>
```

İndidə gəlin list daxilindəki start funksiyamızdan istifadə edək

```
>>> date.start()

 ++++++++++++++
+  Program running    +
+                         +
+                         +
+                         +
++++++++++++++

Salam,proqrama xoş gəlmisiniz.Proqramın açar sözləri,
istifadə qaydaları aşağıda göstərilmişdir.
```

[version],[start],[name],[about]

Proqram terminalından
--> about yazaraq,proqram haqqında geniş məlumat ala bilərsiniz.

write key -->

Bura qədər modulumuzu çağırmağı öyrəndik.Hal-hazırda biz şəxsi modulumuzu yazırıq.

Əgər biz funksiyamızı sonda start() bağlasaydıq import date verən kimi modul artıq def start() funksiyasını aktiv edəcəkdi.

Kodlarımıza biraz əlavələr edək

```
import sys
def about():
    about="""
write key -->
[create..creating file]
[stop...stopping program and exit]
"""
    print(about)

def name():
    name='panda program'
    print(name)

def version():
    version='version 0.1'
    print(version)
def start():
    print("""
++++++++++++++
+  Program running   +
+                          +
+                          +
```

```
+                              +
++++++++++++++
Salam,proqrama xoş gəlmisiniz.Proqramın açar sözləri,
istifadə qaydaları aşağıda göstərilmişdir.

[version],[start],[name],[about]

Proqram terminalından
--> açarsözü yazaraq,proqrama başlaya bilərsiniz

""")

 while True:

        i=input('write key --> ')

        if i=='create':
            v=input('file name:')
            k=input('write file mode : ')
            print('{} file opened'.format(v))
            f=open(v+'.txt',k)
        elif i=='stop':
            print('[-] stopping..')
            sys.exit()
        elif i=='about':
            print(about())

        else:
            print('[-] unknown key...')
```

```
>>> dir(date)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'about', 'name', 'start', 'sys', 'version']
>>>
```

yazaraq,növbəti istifadə edəcəyimiz funksiyaların listəsinə baxırıq. Gördüyünüz kimi 'about', 'name', 'start','version' kimi funksiyalarımız mövcuddur

Python əmr sətrindən

```
>>> date.version()
version 0.1
>>>
```

yazaraq versionu haqqında məlumat əldə edirik.Eyni qayda ilə

....

```
>>> date.name()
panda program
>>> date.about()

write key -->
[create..creating file]
[stop...stopping program and exit]



>>> date.start()

++++++++++++++
+  Program running    +
+                     +
+                     +
+                     +
++++++++++++++

Salam,proqrama xoş gəlmisiniz.Proqramın açar sözləri,
istifadə qaydaları aşağıda göstərilmişdir.

[version],[start],[name],[about]

Proqram terminalından
--> about yazaraq,proqram haqqında geniş məlumat ala bilərsiniz.
```

```
write key --> about

write key -->
[create..creating file]
[stop...stopping program and exit]



None
write key --> name
[-] unknown key...
write key --> create
file name:newfile
write file mode : a
newfile file opened
write key -->
```

....

Və görünüş həmdə funksianal baxımdan modulumuz çalışır.
Kodları bütövlükdə rahat görünüş üçün
https://pastebin.ubuntu.com/23421299/ ünvana baxa bilərsiniz

from modul_adı import metod yolunu keçdiyimiz üçün

```
>>> from date import about
>>> about()





write key -->
[create..creating file]
[stop...stopping program and exit]
>>> version()
version 0.1
>>> start()
```

```
++++++++++++++
+  Program running    +
+                            +
+                            +
+                            +
++++++++++++++
```

Salam,proqrama xoş gəlmisiniz.Proqramın açar sözləri,
istifadə qaydaları aşağıda göstərilmişdir.

[version],[start],[name],[about]

Proqram terminalından
--> about yazaraq,proqram haqqında geniş məlumat ala bilərsiniz.

write key -->

kimi yazaraq istifadə edə bilərik.Hər halda modulların necə yazıldığını az-çox başa düşmüsünüz düşünürəm.Kodlarımız arasına bəzi əlavələr də edərək metodlarımıza argumentlər verərək də istifadə edə bilərik. Beləki os modulunun içində olan system funksiyasının aldığı xterm argumenti buna misal ola bilər.Biz bu argumentləri kodlarımıza əlavə edib onlardan istifadə yollarına baxaq

Kodlarımıza

```
def funk(quote):
    i='{} ədədinin qüvvəti {}'
    print(i.format(quote,pow(quote,2)))
```

əlavə edərək bütünlükdə kodlarımız

```
import sys
```

```
def about():
    about="""
write key -->
[create..creating file]
[stop...stopping program and exit]


"""
    print(about)
def funk(quote):
    i='{} ədədinin qüvvəti {}'
    print(i.format(quote,pow(quote,2)))  #əlavə etdiyimiz yeni funksiya

def name():
    name='panda program'
    print(name)

def version():
    version='version 0.1'
    print(version)
def start():
    print("""
++++++++++++++
+  Program running    +
+                          +
+                          +
+                          +
++++++++++++++

Salam,proqrama xoş gəlmisiniz.Proqramın açar sözləri,
istifadə qaydaları aşağıda göstərilmişdir.

[version],[start],[name],[about]

Proqram terminalından
--> about yazaraq,proqram haqqında geniş məlumat ala bilərsiniz.

""")
    while True:

        i=input('write key --> ')
```

```
    if i=='create':
        v=input('file name:')
        k=input('write file mode : ')
        print('{} file opened'.format(v))
        f=open(v+'.txt',k)
    elif i=='stop':
        print('[-] stopping..')
        sys.exit()
    elif i=='about':
        print(about())

    else:
        print('[-] unknown key...')
```

kimi olacaq.Və yaddaşa verib bağlayırıq.Ardından python əmr sətrindən

```
>>> import date
>>> dir(date)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'about', 'funk', 'name', 'start', 'sys', 'version']
>>> date.funk(4)
4 ədədinin qüvvəti 16
>>> from date import funk
>>> funk(6)
6 ədədinin qüvvəti 36
>>>
```

yazaraq modulumuz daxilində olan funk – funksiyasından istifadə etdik.

Skript daxilinə digər funksiyalar da əlavə edib daha da funksianallığını təmin edə bilərik.Bu sizin ideyalarınıza bağlıdır.Məqsədim modulları necə yazılmasını göstərmək idi.Düşünürəm bura qədər çətinliyiniz olmadı.

Əgər çətinlikləriniz olarsa pythonaz@yahoo.com ünvanına,eləcədə https://techazweb.wordpress.com/forum/ ünvanına yazaraq python istifadəçiləri ilə bərabər bu çətinliyi aradan qaldıra bilərik.Çalışın

öyrənmək üçün gizlənməyin nədə gizlətməyin.Qarşılıqlı öyrənmək hər şeydən üstündür.Çünki hər kəs birxətli düşünmür.

Üçüncü şəxs modullar(third-person modules)

Biraz əvvəl modulları iki yerə bölmüşdük.Bizim yaza biləcəyimiz və standart kitabxana modullar.Bunlardan başqa üçüncü şəxs modulları da mövcuddur.Bu moddullar pythonla bərabər gəlmir,bunları fərqli metodlarla sisteminizə qurmaq lazımdır.Bu modullar da python istifadəçiləri tərəfindən yazılıb,pulsuz olaraq paylaşılır.

Yükləmə qaydası

pip-komandası.
Linuxda apt-əmri ilə proqramları yüklədiyimiz kimi pythonun da şəxsi pip -əmri mövcuddur.Hər şeydən əvvəl pip komandası sisteminizdə yüklü olmalıdır.pip komandası da pythonla yazılımış moduldur,skriptdir.

pip yükləmək

terminalı açırıq

```
$ sudo apt-get install python-pip
[sudo] password for user:
Reading package lists... Done
Building dependency tree
Reading state information... Done
python-pip is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
$
```

və paketin yerini isə

```
$ pip -V
pip 1.5.4 from /usr/lib/python2.7/dist-packages (python 2.7)
$
```

yolla təyin edirik.

Biz hal-hazırda python2 üçün yükləməni yerinə yetirdik.

Python3 -üçün isə

```
$ sudo apt-get install python3-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3-pip is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
$
```

və xətasız yükləyirik.Sistemimdə yüklü olduğu üçün biz yuxarıdakı nəticəni aldıq.

Digər yükləmə yolu

```
easy_install pip
```

Windows istifadəçiləri isə

https://pypi.python.org/pypi ünvanına daxil olaraq tar,zip yükləyib,sıxışdırılmış faylı açaraq içərisində olan setup.py – olan qovluq içində terminal açaraq python setup.py install yazaraq yükləyə bilərsiniz.

Paketi yükləmək üçün

```
pip install paket_adı     #python2 üçün

pip3 install paket_adı  # python3 üçün
```

__all__

Ifadə modul daxilindəki funksiyaları bir yerə toplayır

__all__=['funk','name','version','start','about'] ifadəsini kodlarımıza daxil edib

```
>>> import date
>>> dir(date)
['__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', 'about', 'funk', 'name', 'start', 'sys',
'version']
```

yazdıqda,list daxilində __all__ ifadəsini də görürük.Yəni bu ifadə daxilindəki verilənlər istifadə üçün toplanmış verilənlərdir.Bunu modulu yazan şəxs qeyd edir,yəni biz.İfadədən kənarda qalan funksiyalarda çalışa bilir.

__name__

ifadə yazdığımız modulun adını verir

```
>>> import date
>>> dir(date)
['__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', 'about', 'funk', 'name', 'start', 'sys',
'version']
>>> date.__name__
'date'
>>>
```

istifadə edə biləcəyi metodları isə

```
>>> dir(__name__)
```

yazaraq baxa bilərik.gördüyünüz kimi qarşımıza bir string cins metodları

çıxdı.

```
>>> import date
>>> date.__name__.upper()
'DATE'
```

```
if __name__ == '__main__':
```

ifadə

```
>>> print(__name__)
__main__
>>>
```

bizə __main__ metoduna qarşılıq olduğunu çap etdi
Biz def start() ifadəsini silib,proqramımıza sadəcə modul kimi deyil proqram olaraq çalışdırılmasına zəmin yarada bilərik.Bunun üçün
if __name__ == '__main__': ifadəsini

```
if __name__ == '__main__':
    while True:

        i=input('write key --> ')
        if i=='create':
            v=input('file name:')
            k=input('write file mode : ')
            print('{} file opened'.format(v))
            f=open(v+'.txt',k)
        elif i=='stop':
            print('[-] stopping..')
            sys.exit()
        elif i=='about':
            print(about())
        elif i=='version':
            print(version())

        else:
            print('[-] unknown key...')
```

əlavə edərək yaza bilərik.

Bura qədər modullar haqqında səthi məlumat əldə edərək müəyyən biliklərə yiyələndik.

İpython

ipython-u sisteminizə yükləmək üçün

```
sudo apt-get install ipython
sudo apt-get install ipython3
```

ipython terminaldan çalışa bilən bir proqram olub,yükləndiyi zaman özü ilə bərabər NumPy SciPy matplotlib kitabxanalarını da yükləyir.
Proqramı çağırdıqda isə görünüşü

```
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
Type "copyright", "credits" or "license" for more information.

IPython 1.2.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

kimi olacaq.

Bundan başqa ipython üçün qrafik proqram kimi spyder-i yükləyib istifadə edə bilərsiniz.

Terminaldan bir modul çağıraq

```
In [2]: import re

In [3]:
```

və aşağı sətrə xətasız keçdik.

Re modulunun metodlarına dir() funksiyası ilə baxaq.

```
In [3]: dir(re)
Out[3]:
```

qarşımıza y-oxu üzrə metod və funksiyalar çıxdı

match() metodu

metod veriləni ifadə daxilində olub-olmadığını təyin edir.

```
In [4]: i='python is very best language'
In [5]: re.match('python',i)
Out[5]: <_sre.SRE_Match object; span=(0, 6), match='python'>

In [6]:
```

Yuxarıdakı ifadə bizə kəlimənin I-ifadəsi daxilində olduğunu dedi.Eləcədə span(0,6) ifadəsi python sözünün aralıqda olduğunu təsdiqlədi.Yəni ilk 5 sırana python -ifadəsi tutur.(6-cı sıraya qədər)Beləki

```
In [6]: i[0:6]
Out[6]: 'python'

In [7]:
```

Sorğuya çəkərək buna əmin olduq.Və ya

```
In [7]: i[0:5]
Out[7]: 'pytho'
```

In [8]:

Indidə ifadə daxilində olmayan bir kəliməni sorğuya çəkək.

In [10]: print(re.match('rubby',i))
None

In [11]:

Və None -ifadəsini aldıq.None bizə həmin sözün i-veriləni daxilində olmadığını təsdiq edir.

group() metodu

Metod match metodunun ekrana cavabı çap etmədiklərini ,bizə göstərməyə yardımçı olur.Yəni

In [11]: i
Out[11]: 'python is very best language'

In [12]: v=re.match('python',i)

In [13]: v.group()
Out[13]: 'python'

In [14]:

ifadənin var olduğunu bizə göstərdi.Əgər ifadəmiz veriləndə yoxdursa

In [14]: i
Out[14]: 'python is very best language'

In [15]: v=re.match('javaj',i)

In [16]: v.group()

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-16-05dce475f865> in <module>()
----> 1 v.group()
```

AttributeError: 'NoneType' object has no attribute 'group'

In [17]:

kimi xəta alacağıq.

search() metodu

match metodu ifadənin sacədə ilk veriləninə baxırdısa,search() metodu isə bütünlükdə ifadə daxilində axtarış edir.Dilimizə axtarmaq kimi tərcümə olunur.

In [17]: i='Azərbaycanın sənaye şəhərlərindən Gəncə,Sumqayıt,Şirvan,Mingəçevir'

In [18]: v=re.search('Gəncə',i)

In [19]: v.group()
Out[20]: 'Gəncə'

In [21]:

Bu metodları ipython üzərində yazmaqda məqsədimiz,ipython istifadə üçün işinizə yararlı olmasını göstərməkdir.Əslində ipython daxilində sistemlərin proqramlaşdırılmasını,administrativ komandaları əks etdirir.Python shelldən əlaqə yaratdığımız platformanı,eləcədə ipythonda da yarada bilərik.Misal olaraq

In [28]: import subprocess

In [29]: subprocess.call(['ls','-l','/home'])
total 4
drwxr-x--- 56 user user 4096 Noy 5 11:06 user
Out[29]: 0

```
In [30]:
```

və ya

```
In [30]: subprocess.call(['ls','-l'])
```

yazaraq user qovluğunda olanları listələyə bilərik.

```
In [34]:  uname='uname'

In [35]: uname_arg='-a'

In [36]: subprocess.call([uname,uname_arg])
Linux backbox 4.4.0-45-generic #66~14.04.1-Ubuntu SMP Wed Oct 19
15:05:38 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
Out[36]: 0

In [37]:
```

Modullar

re modulu

dir(re) yazaraq metod və funksiyalarını görə bilərik.

match() metodu

```
>>> import re
>>> i='python is very best language'
>>> v=re.match('python',i)
>>> v.group()
'python'
>>>
```

metod sadəcə ifadədə ilk verilənə baxır.

search() metodu

metod bütünlükdə ifadə daxilində axtarış edir.Dilimizə axtarmaq kimi tərcümə olunur.

```
>>> i
'python is very best language'
>>> v=re.search('very',i)
>>> v.group()
'very'
>>>
```

```
list=['algida','dondurma','perl','rubby','c++','java','Ramin']
global list
def modul():
    i=input('axtardığınız sözü yazın--> ')
    if i in list:
        print('{} ifadəsi bazada var'.format(i))
    elif i not in list:
        print('{} ifadəsi bazada yoxdur'.format(i))
modul()
```

```
>>>
axtardığınız sözü yazın--> perl
perl ifadə bazada var
>>>
axtardığınız sözü yazın--> vaqo
vaqo ifadə bazada yoxdur
>>>
```

search metodu ilə əvəz etsək

```
def modul():
    import re
    while True:
        list=['algida','dondurma','perl','rubby','c++','java','Ramin']
```

```
        s=input('ifadəni yazın -> ')
        for i in list:
            v=re.search(s,i)
            if v:
                print('ifadə bazada var',v.group())
            else:
                print('ifadə bazada yoxdur')
modul()
```

findall()   metodu

Metod mətn daxilində bir ifadənin neçə dəfə keçdiyini təyin edir.

```
import re
text="""
There are 10 types of people.
Those who know binary and those who don't.
I said: 'There are 10 types of people.'.
I also said: 'Those who know binary and those who don't.'.
Isn't that joke so funny?! False
This is the left side of...a string with a right side.
"""
print(re.findall('There',text))
```

```
>>>
['There', 'There']
>>>
```

Deməli iki dəfə keçirmiş

Search() metodundan fərqi ondadırki,search metodu sözü tapdıqda digərlərinə keçid etmir.amma findall metodu bütünlükdə mətni başdan ayağa oxuyur və nəticəni çap edir.

Məsələn

```
import re
from urllib.request import urlopen
url=urlopen('http://www.python.org')
i=str(url.read())
v=re.findall('python',i)
if v:
    print(v)
else:
    print('söz tapılmadı')
```

Və ekran görüntüsündə,ünvan daxilində olan bütün python sözlərini çap etdi

Meta xarakterlər

re modulu üçün xarakterlərlə bağlı https://docs.python.org/3/howto/regex.html ünvana baxa bilərsiniz.

. ^ $ * + ? { } [ ] \ | ( )

\d işarəsi 0-dan 9-a qədər ədədləri ifadə edir [0-9].

\D   [^0-9].

\s   [\t\n\r\f\v].

\S   [^ \t\n\r\f\v].

\w   [a-zA-Z0-9_].

\W   [^a-zA-Z0-9_].

Bu bölmədə baxacağımız xarakterlər yuxarıdakılardır.

[] -xarakteri

List daxilində ifadələri qeyd edək.
list=['john','mohn','leon','julion','jon']

```
>>> import re
>>> list=['john','mohn','leon','julion','jon']
>>> for i in list:
	x=re.search('jo[nh]n',i)
	if x:
		print(x.group())


john
>>>
```

.   -nöqtə meta xarakteri

```
import re
list=['john','mohn','leon','julion','jon']
for i in list:
    if re.match('.o',i):
        print(i)

>>>
john
mohn
jon
>>>
```

Bu xarakter hər hansı bir veriləni əvəz edir.Amma xarakter xüsusi işarələrdən sonra gələn ifadələrdə axtarış etmir.Bunu irəlidə ətraflı öyrənəcəyik.(Yəni \n \t sonra gələn ifadələri sorğuya çəkmir)

```
import re
list=['john','mohn','leon','julion','jon']
for i in list:
    if re.match('j.',i):
        print(i)
```

```
john
julion
jon
>>>
```

```
import re
list=['amanda','adams','amazon','alabama','amalia']
for i in list:
    if re.match('.m',i):
        print (I)
```

```
>>>
amanda
amazon
amalia
>>>
```

* -işarəsi meta xarakteri

Xarakter verilənin ifadə daxilində sayından aslı olmayaraq axtarışını edir.Misallardan daha aydın olacaq.

```
import re
list=['eat','meat','missing','meal','mui']
for i in list:
```

```
    if re.search('ea*',i):
        print(i)

>>>
eat
meat
meal
>>>
```

ifadə daxilində ea -keçən və sonu bizə maraqlı olmayan verilənləri axtardı.

```
import re
list=['eal','eaal','eaaal','ealata','eafrodita']
for i in list:
    if re.match('e.*f',i):
        print(i)



>>>
eafrodita
>>>
```

```
import re
list=['ədalət','fəlakət','kəramət','məlahət','ülviyyə']
for i in list:
    if re.match('.*ət',i):
        print(i)



>>>
ədalət
fəlakət
kəramət
məlahət
>>>
```

search() metodu ilə də eyni nəticəni verəcək

```
import re
list=['ədalət','fəlakət','kəramət','məlahət','ülviyyə']
for i in list:
    if re.search('.*ət',i):
        print(i)

>>>
ədalət
fəlakət
kəramət
məlahət
>>>
```

```
import re,os
i=os.listdir(os.getcwd())
for v in i:
    if re.match('.*mp',v):
        print(i)
```

Sizin istifadəçi adınız

```
import re,os
def search():
    qovluq=os.listdir('/home/user/Downloads/')
    for i in qovluq:
        if re.match('.*iso',i):
            print(i)

search()
```

Siz mp3,flv,mp4 və syaza bilərsiniz

```
>>>
Parrot-full-3.2_amd64.iso
>>>
```

.* -xarakteri bizə əvvəli necə başlayır başlasın sonu .iso veriləni olan ifadələri tələb etdik.

+ - xarakteri

xarakter əvvəli nə ilə başlayır başlasın və ya heç bir veriləni olmasın-sonu istifadəçinin verdiyi argumentə uyğun ifadələri axtarır.Misallara baxaq

```
import re
list=['ədalət','fəlakət','kəramət','ət','fət','məlahət','ülviyyə']
for i in list:
    if re.search('.+ət',i):
        print (I)



>>>
ədalət
fəlakət
kəramət
fət
məlahət
>>>
```

```
import re
list=['manqan','qamma','alan','tiran','elite']
for i in list:
    if re.search('.+an',i):
        print(i)




>>>
manqan
alan
tiran
>>>
```

? -xarakteri

İlk əvvəl keçdiyimiz xüsusi simbollar içində * və + işarələri ifadə daxilində bir və ya bir neçə sayda hərfləri axtarış edirdisə, sual (?) işarəsi isə yoxluğu və ya bir sayda olduğu hərfləri belə nəzərə alaraq axtarışa keçir.

```
import re
list=['fl','ful','fuul','fuuul','flang','fuuuto']
for i in list:
    if re.match('fu?l',i):
        print(i)

>>>
fl
ful
flang
>>>
```

```
import re
list=['fl','ful','fuul','fuuul','flang','fuuuto']
for i in list:
    if re.match('fu ?l',i):
        print(i)

>>>
ful
>>>
```

```
import re
list=['fl','ful','fuul','fuuul','flang','fuuuto']
for i in list:
    if re.match('f+?l',i):
        print(i)
```

```
>>>
fl
flang
>>>
```

{} -xarakteri

Ifadə daxilində verilənin sayını şərti olaraq qeyd edib axtarışını verə bilərik.Daha açıq formada bir hərf və ya sayı-sayını bildirərək ifadədə sorğuya çəkə bilərik.Misallara baxaq.

```
import re
list=['s','sa','saa','saat','saaat','slaaat']
for i in list:
    if re.match('sa{3}t',i):
        print(i)
```

```
>>>
saaat
>>>
```

Kodlarımızda sa ilə başlayan və 3-cü verilәndən 3-ədəd və sonu t-ilə bitən ifadəni sorğuya çəkdik.

Xarakter 2- argument ala bilir,verilənin maksimum və minimum sayı

```
import re
list=['s','sa','saa','saat','saaat','slaaat']
for i in list:
    if re.match('sa{1,3}t',i):
        print(i)
```

Minimum 1-sayda
Maksimum 3-sayda olan
verilənlər

```
>>>
saat
saaat
>>>
```

```
import re
list=['fl','ful','fuul','fuuul','flang','fuuuuto']
for i in list:
    if re.match('fu{1,2}l',i):
        print(i)
```

```
>>>
ful
fuul
>>>
```

Sirkumfleks metaxarakteri (^)

xarakter ifadə daxilindəki verilənlərin sadəcə əvvəlini oxuyur.

```
import re
list=['564343','fgdg454','45gfhfh','fghf34hg','fgdgd','rtyuqw']
for i in list:
    if re.search('^[0-9]',i):
        print(i)
```

Verilənlərin əvvəli 0-9 arasında ədədlərdən təşkil olunması.

```
>>>
564343
45gfhfh
>>>
```

Və gördüyümüz kimi biz sadəcə başlanğıcı rəqəmlərdən ibarət olan ifadələri axtarışa verdik.
Yuxarıdakı [0-9] 0 və 0 da daxil olmaqla 9 və doqquz da daxil olmaqla verilənlərin təyini üçün qısaldılmış birləşmədir.Digər qısaldılmış əlifba

birləşməsi də var ki [a-z] və [A-Z] kimi yazılır.[a-z] kiçik hərflərdən ibarət əlifba,[A -Z] isə bütünü böyük hərflərdən ibarət birləşmədir.Bu hər üç xarakterlərdən irəlidə geniş danışacağıq.

```
import re
list=['564343','fgdg454','45gfhfh','fghf34hg','fgdgd','rtyuqw','AzsAy12s']
for i in list:
    if re.search('^[a-z]',i):
        print(i)

>>>
fgdg454
fghf34hg
fgdgd
rtyuqw
>>>
```

Başlanğıcı kiçik hərflərdən ibarət verilənlərin təyini üçün

```
import re
list=['564343','fgdg454','45gfhfh','fghf34hg','fgdgd','rtyuqw','AzsAy12s']
for i in list:
    if re.search('^[A-Z]',i):
        print(i)

>>>
AzsAy12s
>>>
```

Başlanğıcı böyük hərflərdən təşkil olunmuş verilənlərin təyini

$ -xarakteri

Bu işarə isə sirkumfleks işarəsinin əksinə olaraq ifadənin sonuna baxır

```
import re
list=['string','ing','fringle','fring','digh','dog','distri','sdf565']
for i in list:
```

```
    if re.search('ing$',i):
        print(i)
```

```
>>>
string
ing
fring
>>>
```

Yuxarıdakı kodlarda sadəcə sonu ing([ay]-[en]-[dji] -lə bitən verilənləri axtarmaq üçün dollar xarakterini ing -hərf birləşməsinin ön tərəfində yazdıq və bizə həqiqətən nəticədə ing -lə bitən listdən üç ifadəni verdi. Başqa işarələrdə olduğu kimi bu işarəni də digərləri ilə bərabər istifadə etmək şansımız var.

```
import re
list=['string','ing','fringle','fring','digh','dog','distri','sdf565']
for i in list:
    if re.search('.g$',i):
        print(i)
```

```
>>>
string
ing
fring
dog
>>>
```

Yuxarıdakı kodlarımızda isə ('.g$') əvvəli nə olur olsun sonu g- ilə bitən ifadələri sorğuya çəkdik.

(\) -metaxarakter

xarakter xüsusi xarakterləri verilənlər daxilində istifadə etməyə yardım edir.Misal olaraq pul vahidləri ilə bağlı kodlarımızda dollar-xarakterini istifadə etsək ekran çapında işarəni görəcəyik,və ya sadəcə pul vahidlərindən dollar kursunu ayırd etmək istəsək bəs necə olacaq!.Məsələn

```
import re
list=['12$','10$','30€']
for i in list:
    if re.match('[0-9]',i):
        print(i)
```

```
>>>
12$
10$
30€
>>>
```

Əgər

```
import re
list=['12$','10$','30€']
for i in list:
        if re.match('[0-9]+$',i):
                print (i)
```

kimi yazsaq heç bir nəticə əldə etməyəcəyik.Və tərs əyri xətt(\) xarakterindən istifadə edərək

```
import re
list=['12$','10$','30€']
for i in list:
    if re.match('[0-9]+\$',i):
        print(i)
```

```
>>>
12$
10$
>>>
```

Və sadəcə dollar kursu olan ifadələri sorğuya çəkdik.

```
import re
```

```
list=['12$','10$','30€']
for i in list:
    if re.match('[0-9]+\€',i):
        print(i)
>>>
30€
>>>
```

Düz xətt (|) xarakteri

Qarşılaşdırma xarakteridir

```
import re
list=['elidor','freedom','emidor','teodor','dorelfun','dor56fgil','orellio']
for i in list:
    if re.search('or|or',i):
        print(i)


>>>
elidor
emidor
teodor
dorelfun
dor56fgil
orellio
>>>
```

```
import re
list=['elidor','freedom','emidor','teodor','dorelfun','dor56fgil','orellio']
for i in list:
    if re.search('^or|or$',i):
        print(i)


>>>
elidor
```

emidor
teodor
orellio
>>>
Mötərizə xarakteri ()

xarakter verilənləri qruplaşdırır.

```
import re
from urllib.request import urlopen
url="http://www.python.org"
i=urlopen(url)
v='href=".+html">.+</a>'
for k in i:
    n=re.search(v,str(k))
    if n:
        print(n.group())
```

```
import re
from urllib.request import urlopen
url="http://www.asan.gov.az/"
i=urlopen(url)
v="href=.+css"
for k in i:
    n=re.search(v,str(k))
    if n:
        print(n.group())
```

Ünvan üzərində ctrl+u basırıq və inspekt kodlar səhifəsinə daxil olub səhifənin kodlama dilini və type href tiplərini təyin edirik.

və sonu css ilə bitən başlıqları qruplaşdırdır.

group() metodu

adından da məlum olduğu kimi verilənləri toplayaraq qruplaşdırır.metoddan daha əvvəl istifadə etmişik.Misallara baxaq

```
>>> i="Python is very best language"
```

```
>>> v=re.search("(Python) (is) (very) (best) (language)",i)
>>> print(v.group())
Python is very best language
>>>
```
string cinslərində olduğu kimi metod ifadə daxilində verilənlərlə tək-tək əlaqə qura bilir.

```
>>> v.group(0)
'Python is very best language'
>>> v.group(1)
'Python'
>>> v.group(2)
'is'
>>> v.group(4)
'best'
>>>
```

istifadə edə biləcəyi metodlar üçün

dir(v) yaza bilərik.

groups()   metodu

metod ifadələri mötərizə daxilində çap edirik.python3

```
import re
i="Python is very best language"
v=re.search("(Python) (is) (very) (best) (language)",i)
print(v.groups())

>>>
('Python', 'is', 'very', 'best', 'language')
>>> v.groups(1)
('Python', 'is', 'very', 'best', 'language')
```

Hərfi simvollar,xarakterlər

\s

Simvol,daxilində boşluqları olan verilənləri aşkar edir.

```
import re
list=['5 Frank','Oktan 7','5elli']
for i in list:
    x=re.search("[0-9]\\s[A-Za-z]+",i)
    if x:
        print(x.group())
```

```
>>>
5 Frank
>>>
```

```
import re
list=['5 Frank','Oktan 7','5elli']
for i in list:
    x=re.search("[A-Za-z]+\s[0-9]",i)
    if x:
        print(x.group())
```

Sonu rəqəmlə bitən

Bir boşluq

Əvvəli böyük və ya kiçik hərf

```
>>>
Oktan 7
>>>
```

\d -Onluq sayları təyin edən simvol

simvol həmçinin [0-9] -xarakterinin əvəzləyicisidir.Yuxarıdakı kodlarımızda

```
import re
list=['5 Frank','Oktan 7','5elli']
for i in list:
    x=re.search("[A-Za-z]+\s\d",i)
    if x:
        print(x.group())
```

Onluq sayların əvəzləyicisi

kimi yazsaq da eyni nəticəni alacağıq

```
>>>
Oktan 7
>>>
```

\w -simvolu (alphanumeric)

Həm hərflərdən,həm də saylardan(alphanumeric) ibarət verilənləri təyin edən simvoldur.Yəni [A-Za-z0-9_] -xarakteri ilə eynilik təşkil edir.Diqqətlə fikir versənin simvol hətda alt-tire işarəsini də təyin edə bilir.

```
import re
v='124Avt_?*+#2s'
x=re.search('\w*',v)
print (x.group())
```

```
>>>
124Avt_
>>>
```

Növbəti simvollarımız isə yuxarıdakı keçdiklərimizin tam əksi olaraq verilənləri təyin edirik.

\S -simvolu

\s simvolunun əksi olaraq ifadə daxilində boşluk olmayan verilənləri təyin edir.

```
import re
list=['5 Frank','Oktan 7','5elli']
for i in list:
    x=re.search("\d+\S+[A-Za-z]",i)
    if x:
        print(x.group())

>>>
5elli
>>>
```

Sonu böyük yada kiçik hərflərdən ibarət olsun

Boşluq olmasın

Onluq saylarla başlasın

\D -simvolu

Simvol \d -əksinə olaraq onluq saylar olmayan verilənləri təyin edir.

```
import re
i=['23','John','Edge','23_night','12April']
for v in i:
    k=re.search('\D*',v)
    if k:
        print(k.group())


>>>
John
```

```
Edge
>>>
```

\W -simvolu alfanumerik olmayan verilənləri təyin edir.

```
import re
i=['23','John','Edge','23_night','*^!']
for v in i:
    k=re.search('\W*',v)
    if k:
        print(k.group())
```

```
>>>
*^!
>>>
```

compile() metodu

Metod dilimizə tərtib etmək kimi tərcümə olunur.Bura qədər keçdiyimiz simvollar,xarakterləri compile() metodu daxilində toplayıb,digər metodlarla verilənləri sorğuya çəkəcəyik.Metod əsasən iri həcmli kod bloklarında istifadə olunur.Çünki tərtibetmə prinspi çox sürətlidir.

```
import re
list=['2.4ton','4.6kg','5.2ltr','0.1kg','litr12']
tertib=re.compile('[0-9]\.[0-9]+[A-Za-z]')
for i in list:
    v=tertib.search(i)
    if v:
        print(v.group())
```

```
>>>
2.4t
4.6k
5.2l
0.1k
```

```
>>>
```

Yazdığımız kodları izah edək.re.compile('[0-9]\.[0-9]+[A-Za-z]') ifadəsində

ilk onluq sayla başlasın,daha sonra nöqtə işarəsini təyin etmək üçün tərs əyri xətdən istifadə etdik və nöqtədən sonra yenə onluq say olsun və sonu böyük yada kiçik hərflərdən təşkil olunsun

```
import re
list=['java_3.1','python.4','oracle_2.6','nagios_1.2','litr12']
tertib=re.compile('[a-z_]+[0-9]\.[0-9]')
for i in list:
    v=tertib.search(i)
    if v:
        print(v.group())
```

Xüsusi işarə- nöqtədən qaçmaq üçün tərs əyri xətt

Onluq saylar \d -yaza bilərik

Alfanumerik verilənlər \w -yaza bilərik

```
>>>
java_3.1
oracle_2.6
nagios_1.2
>>>
```

Kodlarımıza izah verək.İlk alfanumerik verilənlər,daha sonra saylar və nöqtə təkrar saylar olan verilənləri sorğuya çəkdik.

Və ya

```
import re
list=['java_3.1','python.4','oracle_2.6','nagios_1.2','litr12']
tertib=re.compile('\w+\d+\.+\d')
for i in list:
    v=tertib.search(i)
    if v:
        print(v.group())
```

```
>>>
java_3.1
oracle_2.6
nagios_1.2
>>>
```

Bu tərtib metodunun bir neçə argumenti var.Beləki uzun bir mətn daxilində əgər bir kiçik hərflə başlayan ifadə axtarırsınızsa sizə yalnız kiçik hərflə yazılan verilən tapılacaq.Amma biz istəyə bilərikki verdiyimiz ifadənin həm kiçiyini həm də böyüyü sorğuya çəkilərək aşkar olsun.Bunun üçün compile metodunun re.IGNORECASE və ya re.I -argumenti var.

re.IGNORECASE və ya re.I

```
import re
text="""
Sockets are the endpoints of a bidirectional communications channel. Sockets may \
communicate within a process, between processes on the same machine, or between \
processes on different continents.Sockets may be implemented over a number of\
different channel types: Unix domain sockets, TCP, UDP, and so on. The socket \
library provides specific classes for handling the common transports as well\
as a generic interface for handling the rest.Sockets have their own vocabulary
"""
tertib=re.compile('socket',re.IGNORECASE)
print(tertib.findall(text))
```

```
>>>
['Socket', 'Socket', 'Socket', 'socket', 'socket', 'Socket']
>>>
```

re.DOTALL veya re.S

Yadınızdadırsa nöqtə (.)-xarakteri bəhsində qeyd etdikki bu xarakter xüsusi işarələrdən sonra gələn ifadələri oxumur.Məsələn \n -sonra gələn hər hansısa bir ifadəyə qətiyyən baxmayacaq.Misal üçün

```
import re
i='Hello Python,\nPython'
print(re.search('Python.*',i).group())
```

```
>>>
Python,
>>>
```

Və gördüyümüz kimi sadəcə birinci sətrə baxdı.

re.S argumenti bu zaman köməyimizə çatacaq.

```
import re
i='Hello Python,\nPython'
tertib=re.compile('Python.*',re.DOTALL)
v=tertib.search(i)
if v:
    print(v.group())
```

```
>>>
Python,
Python
>>>
```

sub() metodu

Metod ifadə daxilində verilənləri -verdiyimiz argumentlə dəyişə bilir.Əgər bir ifadə daxilində bütünlüklə böyük hərflərdən təşkil olunmuş sözü kiçik hərflərdən təşkil edə bilərik.Və ya veriləni tamam başqa verilənlə əvəz edə bilərik.

Dəyişəcəyimiz ifadə

```
import re
i="Marpol-73 qanun , Beynəlxalq dəniz təşkilatı tərəfindən qəbul olunmuş\
   müxtəlif sularda dənizin çirkləndirilməsinə dair qanun'dır.Qanun\
   növbəti dəyişikliyini 78-ci ildə edərək MARPOL-78 adı ilə tanınır."

v=re.compile('qanun',re.I)

print(v.sub('konvensiya',i))
```

Böyük və kiçik hərflərlə yazılanları da nəzərə alsın

```
>>>
Marpol-73 konvensiya , Beynəlxalq dəniz təşkilatı tərəfindən qəbul olunmuş   müxtəlif sularda dənizin çirkləndirilməsinə dair konvensiya'dır.konvensiya   növbəti dəyişikliyini 78-ci ildə edərək MARPOL-78 adı ilə tanınır.
>>>
```

Yuxarıdakı mətndə re.İGNORECASE argumentindən istifadə edərək ifadələr arasında böyük və kiçik qanun sözlərini təyin edib konvensiya sözü ilə əvəz etməyini tələb etdik.Əvəzetməni sub() metodu yerinə yetirdi.

os modulu

import os deyərək modulu çağıra bilərik.
Və ya

from os import*

eləcədə

from os import metod_və ya funksiya

dir(os) yazaraq qarşımıza sıralanan metod və funksiyalara tək-tək baxaq.

```
>>> import os
>>> os.name
'posix'
>>>
```

Əməliyyat sistemimiz Gnu\Linux olduğu üçün

sep argumenti

```
>>> os.sep
'/'
>>>
```

Yuxarıdakı nəticəni əldə edirik.Amma Windows əməliyyat sistemində isə

```
>>> os.sep
'\\'

>>>
```

kimi nəticə alarıq.

Argument fayllar qovluqlar arasında ana qovluqlarla seçilməsini təyin edir.Fərqləndirir.

```
>>> i='nihad','tərlan'

>>> os.sep.join(i)

'nihad/tərlan'

>>>
```

və ifadələr arasına sep-öz işarəsini yerləşdirərək iki ifadəni ayırdı.

os.getcwd() funksiyası

Funksiya,hansı qovluq altında olduğumuz haqqında məlumat verir.

```
>>> from os import getcwd

>>> getcwd()

'/home/panda'

>>>
```

Windows əməliyyat sistemində isə

```
>>> from os import getcwd

>>> getcwd()

'C:\\Documents and Settings\\panda '
```

os.chdir() funksiyası

Funksiya,təyinatını verdiyimiz qovluğa keçid etməyə kömək edir.Necə

linux sistemlərində

panda@backbox:~$ cd /tmp

yazaraq keçid ediriksə,funksiya vasitəsilə

```
>>> from os import chdir
>>> chdir('/home/user')
>>> getcwd()
'/home/user'
>>>
```

chdir('/home/user') – qovluğuna keçid etdik,daha sonra getcwd()-funksiyası ilə hansı qovluqda olduğumuzu test etdik.

os.listdir() funksiyası

Funksiya olduğumuz qovluqdakı fayl və qovluqları göstərir.Linuxda dir və ls funksiyalarını əvəzedicisidir.(Windows-dir əmri,linuxda dir və ls əmri)

Eləcədə təyinatla verdiyimiz qovluğu oxuya bilərik.

```
>>> from os import*
>>> os.listdir()
>>> os.listdir('/home/user')
```

və ya

```
>>> os.listdir('/tmp')
```

və ya

>>> os.listdir('.')

Bir qovluğ geri isə

>>> os.listdir('..')

['user']

Linuxda $ cd .. əmri

>>>

os.curdir() funksiyası

Funksiya əməliyyat sisteminin ana qovluğunda olanları bizə göstərir.

>>> os.listdir(os.curdir)

os.pardir()

Funksiya bir geri qovluğu göstərir,eləcədə os.listdir('..') -ifadəsilə eyni işi yerinə yetirir.Yəni

>>> os.listdir('..')

['user']

>>> os.listdir(os.pardir)

['user']

>>>

Hər iki funksiya ilə eyni nəticəni aldıq.

os.startfile() xüsusi funksiyası

Funksiya yalnız Windows-sistemində çalışır.

Bir faylı ona uyğun proqramla aça bilərsiniz.

```
>>> os.startfile('pythonaz.pdf')
```

Pythonaz odt faylını pdf-formatında açacaqdır.

os.mkdir() funksiyası

Funksiya sisteminizdə qovluq açmağa yardım edir.Eləcədə qovluq yolunu göstərərək,qovluğunuzu istiqamətləndirdiyiniz qovluqlarda aça bilərsiniz.

Qeyd edimki Windows-əməliyyat sistemində funksiya ilə qovluq yaradmaq istəsəniz CON -adlı qovluq yaradmayın,çünki xəta ilə qarşılaşacaqsınız.CON adının Bill Geytslə bağlı hekayəsi olduğu üçün bu adla qovluq yaradılmasını sistemdə qadağan edib.Digər yollarla yaratmaq olur.Linuxda isə istənilən adla qovluq yarada bilərsiniz.Qovluq açanda ehtiyatlı olun,çünki olduğunuz ana platformada olan qovluqları zədələyə bilərsiniz,əvvəl platformada olduğunuz qovluqların adlarına baxın və siz fərqli bir qovluq yaradın

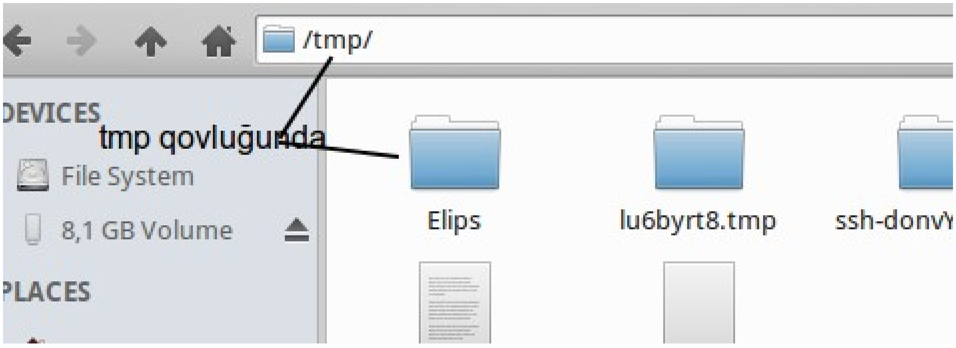
```
>>> os.mkdir('CON')
>>>
```

və aşağı sətrə xətasız keçid etdik.



Və ya istiqamət verərək

```
>>> os.mkdir('/tmp/Elips')
>>>
```



os.makedirs() funksiyası

mkdirs() funksiyası,ilk istiqamətdə yalnız bir ədəd qovluq yarada bilir.Amma makedirs() funksiyası birdən artıq qovluqlar yaratma imkanına malikdir.Yəni
biz '/home/user/qovluq/CON' -yazıb mkdirs() funksiyası ilə CON qovluğunu yarada bilmərik.Çünki qovluq-adlı qovluğumuz olmadığı üçün ardıcıl olaraq iki qovluq yarada bilməz,bu qovluğun sistemdə varolması ehtiyac var.Amma makedirs() funksiyasında yuxarıdakı istiqaməti versəniz asanlıqla nəticəni xətasız əldə edəcəksiniz.Məsələn

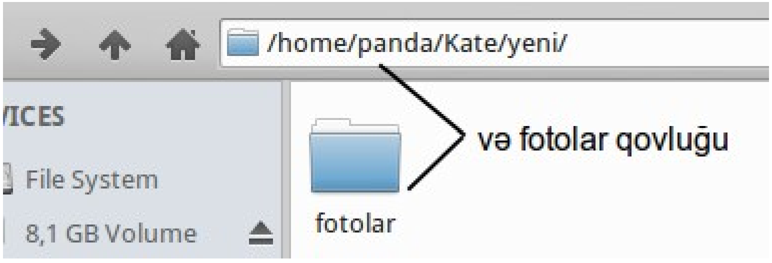
```
>>> os.makedirs('/home/panda/CON/yeni')
>>>
```

yazaraq əvvəl açdığımız CON -qovluğu altında yeni adlı qovluq yaratdıq.

Və ya

```
>>> os.makedirs('/home/panda/Kate/yeni/fotolar')
>>>
```

yazaraq Kate-yeni-fotolar qovluqları yarada bildik.



os.rename() funksiyası

Funksiya, mövcud qovluğun adını digəri ilə əvəz etməyə qadirdir.
Yəni os.rename('köhnə_qovluq','yeniqovluq_adı'
Yuxarıda yaratdığımız fotolar qovluğunu mp3 olaraq dəyişək.

```
>>> os.rename('/home/panda/Kate/yeni/fotolar','mp3')
>>>
```

və aşağı sətrə xətasız keçdik.Qovluğumuzun adı dəyişərək mp3 oldu
Qeyd edimki ad dəyişmədən sonra qovluq olduğu qovluq xətdindən çıxaraq ana qovluq altına keçid edir.Yəni home/user -qovluğu altına keçid edir.Çünki biz qovluğun adını dəyişəndə istiqamət vermədik

os.replace() funksiyası

Funksiya rename() ilə eyni işi görür.Əgər verdiyiniz adla qovluq varsa,içəriyi silinərək yenilənəcək.

os.rmdir() funksiyası

Funksiya qovluğu silmək üçün istifadə olunur.Amma əgər qovluq daxili boş deyilsə o zaman xəta alacaqsınız.Funksiya sadəcə içi boş olan qovluqları kənarlaşdırır,silir.İstifadə edərkən ehtiyatlı olun,çünki silərkən heç bir sual vermir.

```
>>> os.rmdir('/home/panda/Kate/yeni/mp3')
>>>
```

və
mp3 -qovluğunu sildik.Amma Kate qovluğunu silmək istəsək

```
>>> os.rmdir('/home/panda/Kate')
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    os.rmdir('/home/panda/Kate')
OSError: [Errno 39] Directory not empty: '/home/panda/Kate'
>>>
```

xəta ilə qarşılaşacağıq.Xətada deyildiyi kimi Kate qovluğu boş deyil(
Directory not empty).

Bunun üçün python bizə removedirs() funksiyasını təklif edir.

removedirs() funksiyası

```
>>> os.removedirs('/home/panda/Kate/yeni')
>>>
```

və Kate eləcədə qovluq altında olan yeni qovluğu ilə bərabər bütün qovluqlar silindi.

os.stat() funksiyası

Funksiya qovluqlar haqqında məlumat toplamağı icra edir.
Eləcədə funksiya daxilində metodlar istifadə edərək qovluq haqqında tək-tək məlumat toplayır.

```
>>> os.stat('CON')
os.stat_result(st_mode=16893, st_ino=19792179, st_dev=2049,
st_nlink=3, st_uid=1000, st_gid=1000, st_size=4096,
st_atime=1478424104, st_mtime=1478424803, st_ctime=1478424803)
>>>
```

Yuxarıdakı nəticlərdə qovluğun

st_atime   -qovluğa son giriş tarixi
st_ctime   -qovluğun yaranma tarixi
st_mtime -qovluğun dəyişdirilmə tarixi
st_size   -qovluğun həcmi

göstərilmişdir.

Eləcədə ala bildiyi metodlar

```
>>> import os
>>> dir(os.stat('CON'))
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__',
'__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'count', 'index', 'n_fields', 'n_sequence_fields',
'n_unnamed_fields', 'st_atime', 'st_atime_ns', 'st_blksize', 'st_blocks',
'st_ctime', 'st_ctime_ns', 'st_dev', 'st_gid', 'st_ino', 'st_mode', 'st_mtime',
'st_mtime_ns', 'st_nlink', 'st_rdev', 'st_size', 'st_uid']
>>>
```

haqqında məlumat toplaya bilərik.

Istifadə qaydaları isə

```
>>> i=os.stat('CON')
>>> i.st_ctime  #metodu yalnız Windows əməliyyat sistemində məlumat verir.

>>> i.st_atime
```

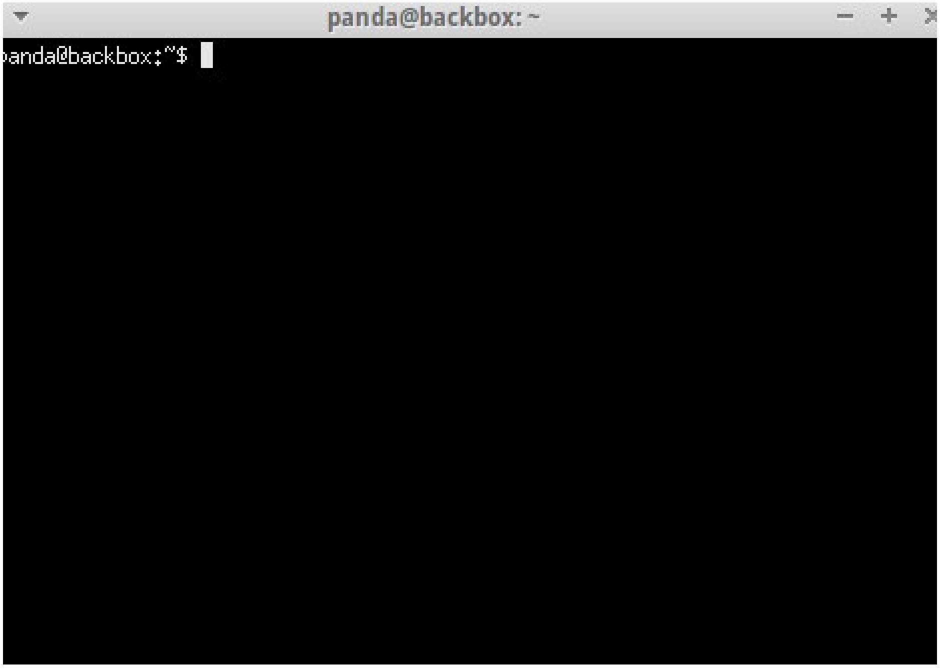
os.system() funksiyası

Funksiya haqqında sabit rəy vermək olmur,çünki funksiya bir çox

metodlar alır.
Misal olaraq

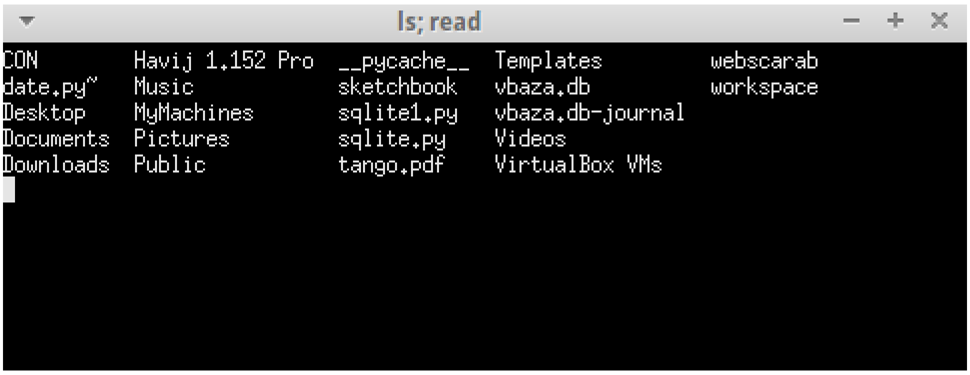
>>> os.system('xterm')

yazaraq qarımıza terminal açılır.



Eləcədə açılan terminala əmrlər verərək

>>> os.system("xterm -e 'ls; read'")

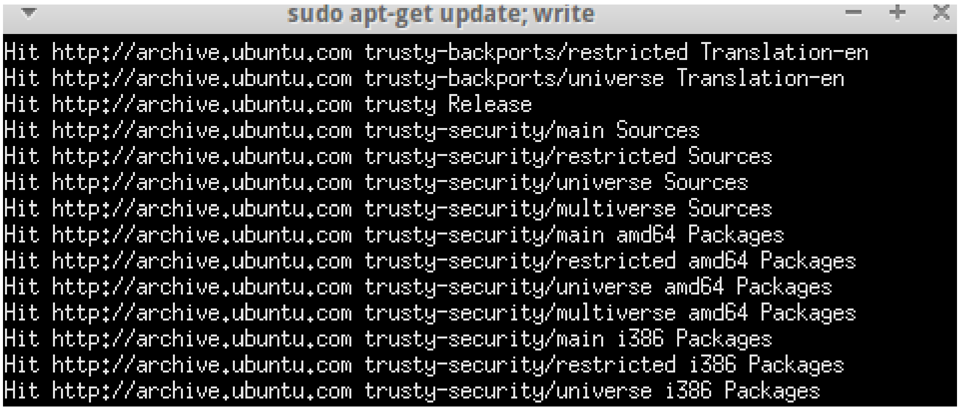


xterm daxilindən home/user qovluğunu oxuya bilərik.
Və ya sistemi update etmək üçün

>>> os.system("xterm -e 'sudo apt-get update; write'")

yazaraq terminala keçid edə bilərik

Ls -əmrində read istifadə etdik,amma update üçün write-yazdıq.



upgrade üçün

>>> os.system("xterm -e 'sudo apt-get dist-upgrade; write'")

yazacağıq.

Sadə bir proqram yazaq.

```
import os,sys  #sistemdən çıxmaq üçün sys modulunu çağırırıq.
print("""
---------------------------------------------------
+ Sisteminizi yüksəltmək üçün          +
+           sudo apt-get update             +
+           sudo apt-get dist-upgrade    +
+ Proqramlara keçid üçün                  +
+            sqlite3                                 +
+            python və ya python3           +
+            idle və ya idle3                      +
+ Sistemdən çıxmaq üçün :  exit         +
+---------------------------------------------------+
""")
def term():
    while True:
        i=input('write command promt >>> ')
        k=input('write or read >>> ')
        v=os.system("xterm -e '{};{}'".format(i,k))
        if i=='exit':
            sys.exit()
        else:
            print(v)
term()



>>>

---------------------------------------------------
+ Sisteminizi yüksəltmək üçün          +
+           sudo apt-get update             +
+           sudo apt-get dist-upgrade    +
+ Proqramlara keçid üçün                  +
+            sqlite3                                 +
+            python və ya python3           +
+            idle və ya idle3                      +
+ Sistemdən çıxmaq üçün :  exit         +
+---------------------------------------------------+

write command promt >>> ls -l
```

```
write or read >>> read
0
write command promt >>>
```

urandom() funksiyası

Funksiya bayt həcmi yaradır.

```
>>> os.urandom(21)
b'L\xb1\x1e\x03\xc9X\x86s\x0fl\xe9\xd9\x1b\xfc\xa0vYCu\x85\xf2'
>>>
```

environ() funksiyası

Funksiya sistemdə olan qruplar,istifadəçilər və s haqqında məlumat verir.

```
>>> os.environ['HOME']
'/home/panda'
>>>

>>> os.environ['USER']
'panda'
>>>
```

Platformalardan asılı olaraq açar sözlər dəyişir.

path funksiyası

funksiya daxilində metodlarla bərabər işlədilir.

abspath() metodu

Faylın yolunu təyin edir

```
>>> os.path.abspath('untitled.py')
'/home/panda/untitled.py'
>>> os.path.abspath('CON')
'/home/panda/CON'
>>>
```

dirname() metodu

Metod faylın qovluqlar yolunu təyin edir.

```
>>> os.path.dirname('untitled.py')
'/home/panda/Desktop'
>>>
```

exists()   metodu

Metod bir qovluq və ya faylın var olub-olmamasını təyin edir.

```
>>> os.path.exists('/home/panda/untitled.py')
False
```

faylımız Desktop-da olduğu üçün False aldıq

amma

```
>>> os.path.exists('/home/panda/Desktop/untitled.py')
True
>>>
```

True verdi.

expanduser() metodu

Metod Sistem istifadəçisiniz hansı qovluqlar altında olduğunu təyin edir.

```
>>> os.path.expanduser('~')
'/home/panda'
```

isdir() metodu

Metod veriləni test edərək qovluq olub-olmadığını yoxlayır.

```
>>> os.path.isdir('/home/panda/CON')
True
>>>
```

Doğru isə -True
Yanlış isə -False

isfile() metodu

Metod,verilənin bir fayl olub-olmadığını test edir.

Bir qovluqdur

```
>>> os.path.isfile('/home/panda/CON')
False
>>> os.path.isfile('/home/panda/Desktop/untitled.py')
True
>>>
```

Bir fayldır

join() metodu

metod ardıcıllığı gözləyərək qovluqlar yaradır.

```
>>> os.path.join('qov1','qov2','qov3')
'qov1/qov2/qov3'
>>>
```

split()   metodu

Metod sonuncu verilən qovluğu bir addım geridəki qovluğdan ayırır.

```
>>> os.path.split('/home/panda/CON')
('/home/panda', 'CON')
>>>
```

splitext()   metodu

metod,faylı-sonluğundan ayırır.

```
>>> os.path.splitext('/home/panda/Desktop/untitled.py')
('/home/panda/Desktop/untitled', '.py')
>>>
```



+
Python

Verilənlərin bazası

İlk əvvəl bunu pythonla əlaqələndirməmişdən öncə verilənlərin bazası haqqında giriş etmək istəyirəm.Daha sonra bunu pythonla yerinə yetirəcəyik.Bu baza bu gün bir çox yerlərdə istifadə olunmaqdadır.Bank sistemləri,maliyəçilik sektorlarında,çoxlu sayda işçilərdən ibarət bir qrum və ya təşkilatlar,sosial şəbəkələr və sairə.Bu gün bir çoxunu misal çəkə bilərik.Hər birinin kökü SQL(Structured Query Language) -ə dayanır.Misal olaraq bu gün rəqabətdə Oracle şirkətini deyə bilərik.Nəticə etibarı ilə bir-birinə bənzəyirlər,yəni kökü SQL-ə dayanır.

SQL nədir?

Təsəvvür edin bir sosial şəbəkəyə qeydiyyatdan keçirsiniz.Və sizin kimi milyonlarca şəxslər var.Bütün bunları bir bazada saxlamaq üçün verilənlərin bazası adı altında cədvəl yaradılır.SQL -in öz sabit ifadələri var.Hər şeydən əvvəl pythonla bu bazanı sorğuya çəkməmişdən əvvəl

SQL haqqında minimal bilikləriniz olmalıdır.Yoxsa çətinliklə qarşılaşacaqsınız.Biz bir neçəsinə nəzər yetirək.Daha sonra sqlite3 istifadə edərək pythonla əlaqələndirək

MySQL üçün

Sisteminizə aşağıdakı paketləri yükləyin

```
sudo pip3 install mysqlclient
sudo pip3 install pymysql
```

Daha sonra MySQL -i yükləyirik

```
sudo apt-get update
sudo apt-get install mysql-server
sudo mysql_secure_installation
sudo mysql_install_db
```

Daha sonra versiasını test edirik.

```
$ mysql --version
mysql  Ver 14.14 Distrib 5.5.53, for debian-linux-gnu (x86_64) using readline 6.3
```

Sistemdə MySQL- servisini test edirik.

```
$ service mysql status
mysql start/running, process 1430
```

və ardından root haqlarından istifadəsini test edirik.

```
$ mysqladmin -p -u root version

Enter password:
mysqladmin  Ver 8.42 Distrib 5.5.53, for debian-linux-gnu on x86_64
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.
```

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

```
Server version          5.5.53-0ubuntu0.14.04.1
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /var/run/mysqld/mysqld.sock
Uptime:                 8 hours 22 min 24 sec

Threads: 1  Questions: 140  Slow queries: 0  Opens: 48  Flush tables: 1
Open tables: 41  Queries per second avg: 0.004
```

....

```
sudo apt-get install sqlite3
sudo pip3 install sqlite3       #python üçün
```

modulu test edirik.

```
$ python3
Python 3.4.3 (default, Sep 14 2016, 12:36:27)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sqlite3
>>>
```

aşağı sətrə xətasız keçid etdiksə deməli modul artıq yüklüdür.

Versionu test edirik.

```
>>> sqlite3.version
'2.6.0'
>>>
```

İstifadə edə biləcəyi metod və funksiyalar

>>> dir(sqlite3)

yazaraq list daxilindəki funksiya və metodları görə bilərik.

SQLite3 + python

Modul olduğundan import ilə çağrılır.

Idle3 üzərindən yeni fayl açıram və sqlite.py qeyd edərək kodlarımızı yazmağa başlayaq

bazanın yaradılmazı

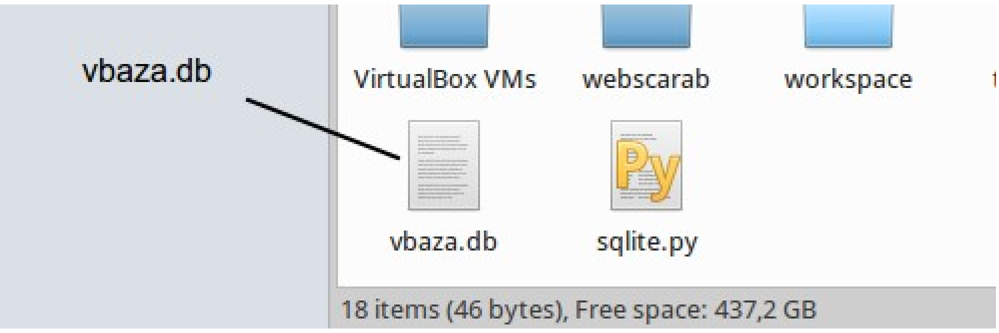
fayl daxilinə

import sqlite3

vb=sqlite3.connect("vbaza.db")

kodları yazaraq proqramı çalışdırıram.

Və aşağı sətrə xətasız keçid edirik,eləcədə python-shell -in çalışdığı qovluğu açaraq bazanın yarandığına əmin olaq.



Bazanın ünvanını yönəldərək də yarada bilərsiniz

vb=sqlite3.connect("/home/user/vbaza.db")

Kodlarımız arasında olan connect funksiyası bazanın yaradılmasında rol oynadı.

Hal-hazırda bazamız boşdur

Cursor,Table and Column

Kursor

Kursor,baza ilə əlaqəni yaradır,yəni verilənləri əlavə etmək,silmək,dəyişikliklər üçün mütləq ilk kursoru yaradmalıyıq.

Kodlarımız arasına

```
import sqlite3
vb=sqlite3.connect("vbaza.db")
crs=vb.cursor()
crs.execute("""CREATE TABLE tabel (tabel_id,tabel_ad,tabel_maas)""")
```

kodunu əlavə edərək (qırmızı boyanmış) proqramı çalışdırırıq.Və aşağı sətrə xətasız keçid edirik.

Biz tabel adında cədvəl yaratdıq,və onun daxilində şəxsi id şəxsi adı və şəxsin maaş sütununu qeyd etdik.

vd- ifadəsi şərti olaraq qeyd etdik.siz istəsəniz fərqli dəyər verə bilərsiniz.Eləcədə crs,kursor-a uyğun olar mən crs qeyd etdim

Daha sonra son kodlarımızı bir ifadə(def ifadəsi) altına alaraq

```
import sqlite3
vb=sqlite3.connect("vbaza.db")
crs=vb.cursor()
def tabel():
    crs.execute("""CREATE TABLE tabel (tabel_id,tabel_ad,tabel_maas)""")
```

yazırıq.

Yeni sütun əlavə etmək üçün ALTER TABLE ifadəsindən istifadə edəcəyik.

```
import sqlite3
vb=sqlite3.connect("vbaza.db")
crs=vb.cursor()
def tabel():
    crs.execute("""CREATE TABLE tabel\
    (tabel_id,tabel_ad,tabel_maas)""")
```

```
def alter():
    crs.execute("""ALTER TABLE tabel ADD tabel_sira""")
alter()
```

Yeni əlavə etdiyimiz sütun(column) tabel_sira oldu.

Sonda isə alter() yazaraq funksiyanı aktiv edirik,və aşağı sətrə xətasız keçid etdiyimizdən davam edək;

Bazaya verilən daxil etmə və bazadan verilənləri oxumaq

Bunun üçün “””INSERT INTO cədvəl_adı VALUES (verilənlər)””” ifadəsindən istifadə edəcəyik.

Kodlarımız arasına funksiya daxilində

```
import sqlite3
vb=sqlite3.connect("vbaza.db")
crs=vb.cursor()
def tabel():
    crs.execute("""CREATE TABLE tabel (tabel_id,tabel_ad,tabel_maas)""")
def alter():
    crs.execute("""ALTER TABLE tabel ADD tabel_sira""")
def data():
    crs.execute("""INSERT INTO tabel VALUES (1,"Rəşad Garayev",400,1)""")
    vb.commit() #verilənləri bazanın yaddaşına verir
data()
```

tabel_id	tabel_ad	tabel_maas	tabel_sira
1	Rəşad Garayev	400	1

uyğun olaraq əlavə etdik.

bazadan verilənləri oxuyaq.Bunun üçün “””SELECT * FROM cədvəl””” ifadəsindən istifadə edəcəyik.

İfadə daxilindəki ulduz işarəsi,cədvəldən bütün sütunları oxuyur,çəkir.

```
import sqlite3
vb=sqlite3.connect("vbaza.db")
crs=vb.cursor()
def tabel():
    crs.execute("""CREATE TABLE tabel (tabel_id,tabel_ad,tabel_maas)""")
def alter():
    crs.execute("""ALTER TABLE tabel ADD tabel_sira""")
def data():
    crs.execute("""INSERT INTO tabel VALUES (1,"Rəşad Garayev",400,1)""")
    vb.commit() #verilənləri bazanın yaddaşına verir
def read_data():
    crs.execute("""SELECT * FROM tabel""")
    verilen=crs.fetchall()
    for i in verilen:                                #son əlavələrimiz.
        print(i)
read_data()
```

Ekran görüntüsü

```
>>>
(1, 'Rəşad Garayev', 400, 1)
>>>
```

Verilənləri əlavə etmək

Yuxarıda data() funksiyamız içindəki kodlara dəyişiklik edərək

```
def data():
    crs.execute("""INSERT INTO tabel VALUES (2,"Səmayə Əliyeva",280,2)""")
    crs.execute("""INSERT INTO tabel VALUES (3,"Nazim Məmmədov",350,3)""")
    crs.execute("""INSERT INTO tabel VALUES (4,"Xəyalə Əliyeva",430,4)""")
```

yazıb,kodlarımızın sonunda

```
import sqlite3
vb=sqlite3.connect("vbaza.db")
crs=vb.cursor()
def tabel():
    crs.execute("""CREATE TABLE tabel (tabel_id,tabel_ad,tabel_maas)""")
def alter():
    crs.execute("""ALTER TABLE tabel ADD tabel_sira""")
def data():
    crs.execute("""INSERT INTO tabel VALUES (2,"Səmayə Əliyeva",280,2)""")
    crs.execute("""INSERT INTO tabel VALUES (3,"Nazim Məmmədov",350,3)""")
    crs.execute("""INSERT INTO tabel VALUES (4,"Xəyalə Əliyeva",430,4)""")
    vb.commit() #verilənləri bazanın yaddaşına verir
def read_data():
    crs.execute("""SELECT * FROM tabel""")
    verilen=crs.fetchall()
    for i in verilen:
        print(i)
data()
read_data()
```

data() və read_data() funksiyalarını bağlayaraq aktiv edirik.

Və nəticəmiz

```
>>>
(1, 'Rəşad Garayev', 400, 1)
(2, 'Səmayə Əliyeva', 280, 2)
```

(3, 'Nazim Məmmədov', 350, 3)

(4, 'Xəyalə Əliyeva', 430, 4)

>>>

İlk əvvəldən verilənləri for operatoru daxilində yazdımki irəlidə ifadələr əlavə etdikdə yuxarıdakı görünüşlə verilənləri oxuyaq.

Təkrar sətirlərin aradan qaldırılması (SELECT DISTINCT * FROM cədvəl_adı)

read_data funksiyamız daxilində kodlara dəyişiklik edərək;

```
def read_data():
    crs.execute("""SELECT DISTINCT * FROM tabel""") #yeni ifadəmiz
    verilen=crs.fetchall()
    for i in verilen:
        print(i)
read_data()
```

Bütünlükdə kodlara https://pastebin.ubuntu.com/23426544/ ünvanından baxa bilərsiniz

proqramı çalışdırırıq,və ekran görüntüsü

```
>>>
(1, 'Rəşad Garayev', 400, 1)
(2, 'Səmayə Əliyeva', 280, 2)
(3, 'Nazim Məmmədov', 350, 3)
(4, 'Xəyalə Əliyeva', 430, 4)
>>>
```

ORDER BY ifadəsi

İfadə verilənləri baza daxilində əlifba sırası ilə düzür.Verdiyiniz komandaya uyğun olaraq;məsələn verilənləri id-lərə görə,adlara görə,maaşa görə və sairə;

read_data() funksiyası daxilinə kodlara dəyişiklik edərək

Adlara görə

```
def read_data():
    crs.execute("""SELECT DISTINCT * FROM tabel ORDER BY tabel_ad DESC""")
    verilen=crs.fetchall()
    print('Verilənlər əlifba sırası ilə düzüldü')
    for i in verilen:
        print(i)
read_data()

>>>
Verilənlər əlifba sırası ilə düzüldü
(4, 'Xəyalə Əliyeva', 430, 4)
(2, 'Səmayə Əliyeva', 280, 2)
(1, 'Rəşad Garayev', 400, 1)
(3, 'Nazim Məmmədov', 350, 3)
>>>
```

və ya maaşa görə

Maaşa görə

```
def read_data():
    crs.execute("""SELECT DISTINCT * FROM tabel ORDER BY tabel_maas DESC""")
    verilen=crs.fetchall()
    print('Verilənlər maaşa görə sıralandı')
    for i in verilen:
        print(i)
read_data()

>>>
Verilənlər maaşa görə sıralandı
(4, 'Xəyalə Əliyeva', 430, 4)
(1, 'Rəşad Garayev', 400, 1)
(3, 'Nazim Məmmədov', 350, 3)
(2, 'Səmayə Əliyeva', 280, 2)
```

>>>

Nəticədən gördüyümüz kimi,şəxslərin sıralanması,maaşa görə çoxdan-aza doğru sıralandı.

Bütünlükdə kodlara https://pastebin.ubuntu.com/23426586/ ünvanından baxa bilərsiniz.

WHERE funksiyası

Məsələn siz bir mühasibatçılıq şirkətində işləyirsiniz.Müdir sizə gəlib deyirki,mənə maaşları 350azn-dən aşağı olan işçilərin siyahısını tərtib et.Bu zaman köməyinizə pythonda olan qarşılaşdırma işarələri və sqlite3-ün bizə təklif etdiyi WHERE funksiyası köməyimizə çatacaq.

Eyni qayda ilə read_data funksiyası daxilində

```
def read_data():
    crs.execute("""SELECT DISTINCT * FROM tabel WHERE tabel_maas < 350""")
    verilen=crs.fetchall()
    print('Maaşları 350aznd-dən az olanlar')
    for i in verilen:
        print(i)
read_data()
```

kodlarımıza əlavələrimizi edirik.Və proqramı çalışdırırıq.

```
>>>
Maaşları 350aznd-dən az olanlar
(2, 'Səmayə Əliyeva', 280, 2)
>>>
```

Deməli bazamızda sadəcə bir şəxsin maaşı 350 azndən az olan verilən var.

DELETE FROM funksiyası

Funksiya bazada istənilən veriləni silmək üçün köməyimizə çatacaq.Ən başda id -əlavə etməyimiz bu silmə metodunda bizə əlverişli olacaq.Yəni bir veriləni id-nömrəsinə görə bazadan siləcəyik

Yeni bir def sil() funksiyası yaradaraq;

```
def sil():
    crs.execute("""DELETE FROM tabel WHERE tabel_id == 1""")
    vb.commit()
    print('Şəxs bazadan kənarlaşdırıldı')
sil()
read_data()
```

Kodlarımızı çalışdırırıq.Qeyd edimki bütövlükdə kodlarımızı yazmamaqda məqsədim çox yer tutduğu üçün izah etmək əlverişli olmayacaq.Ona görədə dəyişiklik olan kodları bura yazıram və qeyd etdiyim ünvanlara daxil olub kodları bütünlükdə görə bilərsiniz.Və davam edərək proqramı çalışdırırıq;

```
def sil():
    crs.execute("""DELETE FROM tabel WHERE tabel_id == 1""")
    vb.commit()
    print('Şəxs bazadan kənarlaşdırıldı')
sil()
```

Kodlarımız bütünlükdə

```
import sqlite3
vb=sqlite3.connect("vbaza.db")
crs=vb.cursor()
```

```
def tabel():
    crs.execute("""CREATE TABLE tabel(tabel_id,tabel_ad,tabel_maas)""")
def alter():
    crs.execute("""ALTER TABLE tabel ADD tabel_sira""")
def data():
    crs.execute("""INSERT INTO tabel VALUES (2,"Səmayə Əliyeva",280,2)""")
    crs.execute("""INSERT INTO tabel VALUES (3,"Nazim Məmmədov",350,3)""")
    crs.execute("""INSERT INTO tabel VALUES (4,"Xəyalə Əliyeva",430,4)""")
    vb.commit() #verilənləri bazanın yaddaşına verir
def read_data():
    crs.execute("""SELECT DISTINCT * FROM tabel """)
    verilen=crs.fetchall()
    for i in verilen:
        print(i)
def sil():
    crs.execute("""DELETE FROM tabel WHERE tabel_id == 1""")
    vb.commit()
    print('Şəxs bazadan kənarlaşdırıldı')
read_data()
```

Maaş təyini ilə dəyişiklik etdiyimiz kodu yenidən əvvəlki vəziyyətinə qaytardıq

```
>>>
(2, 'Səmayə Əliyeva', 280, 2)
(3, 'Nazim Məmmədov', 350, 3)
(4, 'Xəyalə Əliyeva', 430, 4)
>>>
```

UPDATE SET funksiyası

Əgər maaş artımı ilə hansısa bir qərar olarsa və ya verilənlər üzərində dəyişiklik etmək istəsəniz bu funksiyadan istifadə edə bilərsiniz.Gəlin funksiyadan istifadə edərək işçilərin maaşını 1.5 dəfə qaldıraq.

Bunun üçün yeni def update() funksiyası yaradırıq.

```
def update():
    crs.execute("""UPDATE tabel SET tabel_maas =tabel_maas*1.5""")
read_data()
print('Maaşlar yüksəldikdən sonra')
update()
read_data()
```

crs.execute -daxilindəki kodlarımız tabel_maaş-ı 1.5 dəfə hasil edərək şəxslərin maaşını artırdı.daha sonra əvvəlki bazadakı verilənləri oxuduq(read_data()).Sonra istifadəçiyə yeni maaşlar haqqında məlumat verib(print('Maaşlar yüksəldikdən sonra')) update() funksiyasını aktiv edərək son bazadakı dəyişiklikləri çap etdirdik.

Bütövlükdə kodlarımız

```
import sqlite3
vb=sqlite3.connect("vbaza.db")
crs=vb.cursor()
def tabel():
    crs.execute("""CREATE TABLE tabel(tabel_id,tabel_ad,tabel_maas)""")
def alter():
    crs.execute("""ALTER TABLE tabel ADD tabel_sira""")
def data():
    crs.execute("""INSERT INTO tabel VALUES (2,"Səmayə Əliyeva",280,2)""")
    crs.execute("""INSERT INTO tabel VALUES (3,"Nazim Məmmədov",350,3)""")
    crs.execute("""INSERT INTO tabel VALUES (4,"Xəyalə Əliyeva",430,4)""")
    vb.commit() #verilənləri bazanın yaddaşına verir
def read_data():
    crs.execute("""SELECT DISTINCT * FROM tabel """)
    verilen=crs.fetchall()
    for i in verilen:
```

```
        print(i)
def sil():
    crs.execute("""DELETE FROM tabel WHERE tabel_id == 1""")
    vb.commit()
    print('Şəxs bazadan kənarlaşdırıldı')
def update():
    crs.execute("""UPDATE tabel SET tabel_maas =tabel_maas*1.5""")
read_data()
print('Maaşlar yüksəldikdən sonra')
update()
read_data()
```

SUM funksiyası

Bazaların daxilində bir çox riyazi funksiyalar çalışır.Bunlardan biri SUM funksiyasına baxacağıq.Bu funksiya vasitəsilə bazada olan şəxslərin maaşlarının toplamını hesablayacağıq.Yəni şirkətin öz işçilərinə ümumi verdiyi maaşın toplamı.

Sum funksiyası üçün def sum_funk() yazaraq

```
def sum_funk():
    crs.execute("""SELECT SUM(tabel_maas) FROM tabel""")
    verilen=crs.fetchall()
    print(verilen)
sum_funk()
```

```
>>>
[(1590.0,)]
>>>
```

Deməli şirkət işçilərinə toplam 1590.0 azn pul verir.

AVG funksiyası

Funksiya bazadakı verilənlərin ortalama dəyərini çıxarır.Bu id-dəyəri,maaşlar və s ola bilər.biz cədvəlimizdə sadəcə 4 -sütun açmışıq,amma vrilənlərin bazası bu deyil,daha böyük bazalara malik şirkətlər var.Sadəcə biz bu gün pythonla v.bazasının istifadəsini öyrənirik.

Funksiyadan istifadə üçün def avg_funk() yazaraq daxilinə kodlarımızı

```
def avg_funk():
    crs.execute("""SELECT AVG(tabel_maas) FROM tabel""")
    verilen=crs.fetchall()
    print(verilen)
avg_funk()
```

əlavə edirik.

Ekran görüntüsü

```
>>>
[(353.3333333333333,)]
>>>
```

dəyərini çap etdi.

MAX və MİN funksiyaları

Bu funksiyalar sütunlarda olan minimum və maksimum dəyərləri çıxarır.

```
def max_funk():
    crs.execute("""SELECT MAX(tabel_maas) FROM tabel""")
    verilen=crs.fetchall()
    print(verilen)
def min_funk():
    crs.execute("""SELECT min(tabel_maas) FROM tabel""")
    verilen=crs.fetchall()
```

```
    print(verilen)
max_funk()
min_funk()
```

```
>>>
[(430,)]
[(280,)]
>>>
```

bazada maksimum dəyər-430

minimum dəyər-280

COUNT funksiyası

SELECT COUNT(*) FROM cədvəl_adı

Funksiya bazadan məlumat süzdürür.Məsələn bazada neçə işçi var,necə nəfər 200 manat maaş alır və sairə.Ulduz işarəsinin nə olduğunu bilirik,yenə təkrar edərək bütün sütunları oxuyur.

```
def count_funk():
    crs.execute("""SELECT COUNT (*) FROM tabel""")
    i=crs.fetchall()
    print(i)
count_funk()
```

```
>>>
[(3,)]
>>>
```

Deməli bazada 3 işçi var.

SELECT name FROM sqlite_master WHERE type = ‘table’ -ifadəsi

ifadə bazadakı cədvəl adlarını süzür.Yəni yuxarıdakı ifadədə,sqlite_master cədvəlindən(standart) name sütunundakı(column) cədvəllərə ait adları çək mənasını ifadə edir.

```
def base():
    crs.execute("""SELECT name FROM sqlite_master WHERE type='table'""")
    i=crs.fetchall()
    print(i)
```

Və pytohn-shell-dən

```
>>> base()
[('tabel',)]
>>>
```

yazaraq bizə tabel adlı-cədvəlin olduğunu verdi.

Sisteminizdə test üçün müvəqqəti baza yarada bilərsiniz,bunun üçün yeni bir fayl açırıq,ardından kodlarımızı yazırıq.

```
import sqlite3
vb=sqlite3.connect(":memory:")
crs=vb.cursor()
```

Və yaddaşa verib çalışdırdıqda aşağı sətrə xətasız keçid edirik.Kodlarımız içində müvəqqəti baza üçün ":memory:" -ifadəsindən istifadə etdik.

AUTO INCREMENT və PRIMARY KEY

Bəzən yazdığımız sütunların avtomatik artmasını,verilən əlavə olunduqda öz mümkün yerini tutmasını istəyirik.Bunun üçün bu funksiyalar işimizə yarayacaq.

```
import sqlite3
vb=sqlite3.connect(":memory:")
```

```
crs=vb.cursor()
def ibook():
    crs.execute("CREATE TABLE books(b_id INTEGER PRIMARY KEY AUTOINCREMENT,b_autor
        TEXT,b_name TEXT)")
    crs.execute("INSERT INTO books (b_autor,b_name) VALUES ('Qurban Said','Əli və Nino')")
    crs.execute("INSERT INTO books (b_autor,b_name) VALUES ('Xalid Hoseyni','Çərpələng uçuran')")
    crs.execute("INSERT INTO books (b_autor,b_name) VALUES ('Q.Markez','Kimyagər')")
    crs.execute("INSERT INTO books (b_autor,b_name) VALUES ('Süleyman Rüstəm','Qaraçı qız')")
    crs.execute("SELECT * FROM books")
    books=crs.fetchall
    print(books)
ibook()
```

```
'''

+---+--------------------+------------------+
+ 1 + Qurban Said          Əli və Nino      +
+---+--------------------+------------------+
+ 2 + Xalid Hoseyni        Çərpələng uçuran +
+---+--------------------+------------------+
+ 3 + Q.Markez             Kimyagər         +
+---+--------------------+------------------+
+ 4 + Süleyman Rüstəm      Qaraçı qız       +
+---+--------------------+------------------+

>>> |
```

LIKE funksiyası

Funksiya bazada verilən 'x' -ifadənin axtarışına çıxır.

```
def oxu():
    crs.execute("""SELECT * FROM tabel WHERE tabel_ad LIKE 'X%'""")
    i=crs.fetchall()
    print(i)

>>> oxu()
[(4, 'Xəyalə Əliyeva', 430, 4)]
>>>
```

'x%'	X -ilə başlayan bütün verilənlər
'%x%'	İçərisində x -olan bütün verilənlər
'_xx%'	2 və 3 xarakteri x-olan bütün verilənlər
%x	X-ilə bitən bütün verilənlər

Yuxarıdakı cədvəldə x- yazdığım x-hərfi deyil,yəni ora istənilən xarakter yazıb bazada axtarışını edə bilərsiniz.

Bura qədər sqlite3-ün lazım olan funksiyalarını öyrəndik.

datetime modulu

```
>>> import datetime
>>> dir(datetime)
['MAXYEAR', 'MINYEAR', '_EPOCH', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'date', 'datetime', 'datetime_CAPI', 'time', 'timedelta', 'timezone', 'tzinfo']
>>> from datetime import datetime
>>> dir(datetime)
['__add__', '__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__', '__ne__', '__new__', '__radd__', '__reduce__', '__reduce_ex__', '__repr__', '__rsub__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', 'astimezone', 'combine', 'ctime', 'date', 'day', 'dst', 'fromordinal', 'fromtimestamp', 'hour', 'isocalendar', 'isoformat', 'isoweekday', 'max', 'microsecond', 'min', 'minute', 'month', 'now', 'replace', 'resolution', 'second', 'strftime', 'strptime', 'time', 'timestamp', 'timetuple', 'timetz', 'today', 'toordinal', 'tzinfo', 'tzname', 'utcfromtimestamp', 'utcnow', 'utcoffset', 'utctimetuple', 'weekday', 'year']
>>>
```

now() funksiyası year,month,day,hour,minute,second

```
>>> import datetime
>>> i=datetime.datetime.now()
>>> i
datetime.datetime(2016, 11, 6, 17, 50, 14, 38096)
>>>

>>> i.year
2016
>>> i.month
11
>>> i.day
6
>>> i.hour
17
>>> i.minute
50
>>> i.second
14
>>>
```

today() funksiyası

```
>>> i=datetime.datetime.today()
>>> i.year
2016
>>> i.month
11
>>> i.minute
```

```
53
>>> i
datetime.datetime(2016, 11, 6, 17, 53, 24, 202422)
>>>
```

ctime() funksiyası

Funksiya tarix və saatı ifadə edir.

```
>>> i=datetime.datetime.now()
>>> zaman=datetime.datetime.ctime(i)
>>> zaman
'Sun Nov  6 17:55:35 2016'
>>>
```

strftime() funksiyası

tarix və saatla bağlı funksiyadır.argumentlər alaraq birbaşa saat,gün,ayı,ili sorğuya çəkə bilərsiniz.

%a   -həftəni ifadə edirik.(ixtisarla)

%A   -həftəni bütünlükdə ifadə edirik.

%b   -ayı ifadə edir.(İxtisarla)

%B   -ayı bütünlükdə ifadə edir.

%c   -tarix və saatı ifadə edir.

%d   -günü sözlə ifadə edir.

%j   -tarixin hansısa bir ildə uyğun gəldiyi gununu ifadə edirik.

%m   -ayı rəqəmlə ifadə edir.

%U   -bir tarixin ilin neçənci həftəsinə uyğun gəldiyini təyin edir.

%y   -İlin son iki rəqəmi

%Y   -ili ifadə edir.

%x   -tarixi bütünlükdə ifadə edir.

%X   -saatı bütünlükdə ifadə edir.

```
>>> i=datetime.datetime.now()
>>> zaman=datetime.datetime.strftime(i,'%A')
>>> zaman
'Sunday'
>>> i=datetime.datetime.now()
>>> zaman=datetime.datetime.strftime(i,'%c')
>>> zaman
'Sun Nov  6 17:59:44 2016'
>>> i=datetime.datetime.now()
>>> zaman=datetime.datetime.strftime(i,'%B')
>>> zaman
'November'
>>>
```

Sistemin lokal zaman məlumatını əldə etmək üçün

```
>>> locale.setlocale(locale.LC_ALL,'')
'LC_CTYPE=en_US.UTF-8;LC_NUMERIC=az_AZ;LC_TIME=az_AZ;LC_COLLATE=en_US.UTF-8;LC_MONETARY=az_AZ;LC_MESSAGES=en_US.UTF-8;LC_PAPER=az_AZ;LC_NAME=az_AZ;LC_ADDRESS=az_AZ;LC_TELEPHONE=az_AZ;LC_MEASUREMENT=az_AZ;LC_IDENTIFICATION=az_AZ'
>>>
```

```
>>> datetime.datetime.strftime(i,'%B')
'noyabr'
>>> datetime.datetime.strftime(i,'%A')
'bazar günü'
>>>
```

Gördüyünüz kimi lokal ünvanı Azərbaycan-üzrə təyin etdikdən sonra ayı həftənin gününü sorğuya çəkdikdə bizə Azəraycanca nəticə verdi.

```
>>> datetime.datetime.strftime(i, '%d %B %Y')
'06 noyabr 2016'
>>>
```

time   modulu

```
>>> import time
>>>
```

aşağı sətrə xətasız keçid edirik.Deməli modul yüklüdür.

gmtime() funksiyası

Hal-hazırki zaman

```
>>> import time
>>> time.gmtime()
time.struct_time(tm_year=2016, tm_mon=11, tm_mday=6, tm_hour=12, tm_min=41, tm_sec=39, tm_wday=6, tm_yday=311, tm_isdst=0)
>>> time.gmtime(12)
time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=0, tm_min=0, tm_sec=12, tm_wday=3, tm_yday=1, tm_isdst=0)
>>>
```

Yuxarıda gördüyünüz nəticədə tm_ lə başlayanlar bir metodlardır.

epoch -zamanın aşlanğıcı

zamanın başlanğıcı bu sahədə epoch olaraq adlandırılır.

```
>>> epoch=time.gmtime()
>>> epoch.tm_year
2016
```

Hal-hazırdakı il

```
>>> epoch.tm_mon
11
```

Neçənci ay

```
>>> epoch.tm_mday
6
>>>
```

Ayın tarixi

time()   funksiyası

Hal-hazırdakı vaxta qədər keçən saniyə miqdarı

```
>>> time.time()
1478436524.027027
>>> time.gmtime(time.time())
time.struct_time(tm_year=2016, tm_mon=11, tm_mday=6, tm_hour=12,
tm_min=48, tm_sec=56, tm_wday=6, tm_yday=311, tm_isdst=0)
>>>
```

localtime()   funksiyası

Funksiya şəbəkəyə bağlandığı üçün dəqiq geo-vaxtı təyin edir.

```
>>> time.localtime()
time.struct_time(tm_year=2016, tm_mon=11, tm_mday=6, tm_hour=16,
tm_min=50, tm_sec=33, tm_wday=6, tm_yday=311, tm_isdst=0)
>>>
```

asctime() funksiyası

Funksiya saat və tarixi görünüşlü çap edir.

```
>>> time.asctime()
```

```
'Sun Nov  6 16:52:27 2016'
>>>
```

Digər funksiyalarla da bərabər istifadə edilə bilinir.

```
>>> time.asctime(time.gmtime())
'Sun Nov  6 12:53:26 2016'
>>>
```

strftime() funksiyası

Funksiya yalnız saat və tarixi çap edir.Argument aldığından aşağıdakı argumentlərə nəzər yetirək.

%a -həftəni ifadə edirik.(ixtisarla)

%A -həftəni bütünlükdə ifadə edirik.

%b -ayı ifadə edir.(İxtisarla)

%B -ayı bütünlükdə ifadə edir.

%c -tarix və saatı ifadə edir.

%d -günü sözlə ifadə edir.

%j -tarixin hansısa bir ildə uyğun gəldiyi gununu ifadə edirik.

%m -ayı rəqəmlə ifadə edir.

%U -bir tarixin ilin neçənci həftəsinə uyğun gəldiyini təyin edir.

%y -İlin son iki rəqəmi

%Y -ili ifadə edir.

%x -tarixi bütünlükdə ifadə edir.

%X -saatı bütünlükdə ifadə edir.

```
>>> time.strftime('%A')
'Sunday'
>>>
```

strptime()   funksiyası

funksiya strftime() funksiyasının aldığı argumentləri qəbul edərək verdiyimiz tarixi çevirir.

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL)
'LC_CTYPE=en_US.UTF-8;LC_NUMERIC=C;LC_TIME=C;LC_COLLATE=C;LC_MONETARY=C;LC_MESSAGES=C;LC_PAPER=C;LC_NAME=C;LC_ADDRESS=C;LC_TELEPHONE=C;LC_MEASUREMENT=C;LC_IDENTIFICATION=C'
>>> i='16 July 2003'
>>> v=time.strptime(i, '%d %B %Y')
>>> v
time.struct_time(tm_year=2003, tm_mon=7, tm_mday=16, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=2, tm_yday=197, tm_isdst=-1)
>>>
```

sleep() funksiyası

Funksiya müddətli vaxtı saxlayaraq proqramı dayandırır.

```
import os,sys,time #sistemdən çıxmaq üçün sys modulunu çağırırıq.
print("""
-------------------------------------------------
+ Sisteminizi yüksəltmək üçün        +
+           sudo apt-get update          +
+           sudo apt-get dist-upgrade  +
+ Proqramlara keçid üçün              +
+            sqlite3                          +
+            python və ya python3       +
+            idle və ya idle3                +
+ Sistemdən çıxmaq üçün :  exit      +
+-----------------------------------------------+
```

```
""")
def term():
    while True:
        i=input('write command promt >>> ')
        k=input('write or read >>> ')
        v=os.system("xterm -e '{};{}'".format(i,k))
        if i=='exit':
            sys.exit()
        else:
              print(v)
time.sleep(15)#15 saniyə sonra çalışmasını təmin edirik
term()
```

random modulu

Modul,təsadüfi rəqəmlərin təşkili üçün alternativ moduldur.

```
>>> import random
>>> dir(random)
```

Yazaraq funksiyalarını görə bilərik.

random() funksiyası

```
>>> random.random()
0.253141076393285
```

```
>>> for i in range(4):
        print(random.random())
```

```
0.7269604890196386
0.37492619677206607
0.28710418332981946
0.6692512619153967
>>> for i in range(4):
	print("{:.5f}".format(random.random()))



0.38977
0.10696
0.26003
0.77310
>>>
```

uniform() funksiyası

Funksiya 2 argument ala bilir.random funksiyası sadəcə 0 və 1 ədədləri arasında təsüdüfi saylar verirsə,uniform funksiyası isə istifadəçinin verdiyi saylar arasında təsadüfi ədədləri verir.

```
>>> random.uniform(0.1,0.8)
0.7875315876696911
>>> random.uniform(0.1,0.8)
0.6686793265636556
>>> random.uniform(0.1,0.8)
0.2221579279705392
>>>
```

randint()   funksiyası

Funksiya kəsirli saylar deyil,tam ədədləri təsadüfi seçir.

```
>>> random.randint(1,4)
1
>>> random.randint(1,4)
3
>>>
```

choice() funksiyası

Funksiya təsadüfi ifadələri seçir.Daha çox str cins ifadələrində istifadə olunur.

```
>>> list=['Vahid','Ruslan','Arzu','Nadir','Fazil','Xəyalə' ,'Mətanət']
>>> random.choice(list)
'Fazil'
>>> random.choice(list)
'Mətanət'
>>>
```

shuffle() funksiyası

Funksiya vasitəsilə list daxilindəki ifadələri təsadüfi qarışdıra bilərsiniz

```
>>> i=['Vahid','Ruslan','Arzu','Nadir','Fazil','Xəyalə' ,'Mətanət']
>>> random.shuffle(i)
>>> i
['Arzu', 'Nadir', 'Mətanət', 'Xəyalə', 'Ruslan', 'Fazil', 'Vahid']
>>>
```

randrange() funksiyası

Funksiya randint funksiyası ilə eyni işi görür.Sadəcə olaraq maksimum verilənin qiyməti nəzərə alınmaz.Yəni biz 2 və 15 aralıqdakı ədədləri təsadüfi göstərsək 15-ə qədər ədədlər nəzərə alınacaq.Minimum dəyər 2-olacaq

```
>>> random.randrange(2,15)
6
>>>
```

sample() funksiyası

Dilimizə nümunə kimi tərcümə olunur.Funksiya list daxilində ifadələrdən biz verdiyimiz dəyərdə nümunə verilənləri alır.

```
>>> i
['Arzu', 'Nadir', 'Mətanət', 'Xəyalə', 'Ruslan', 'Fazil', 'Vahid']
>>> random.sample(i,4)
['Fazil', 'Arzu', 'Mətanət', 'Xəyalə']
>>>
```